НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-сервіс для планування та трекінгу задач»

Виконала:
студентка IV курсу, групи КП-61
Маслюк Уляна Олександрівна _____

Керівник:
Старший викладач кафедри ПЗКС, к.т.н.,
Люшенко Леся Анатоліївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.ф.-м.н., доцент,
Шубенкова Ірина Анатоліївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студентці

Маслюк Уляні Олександрівні

1. Тема проєкту «Веб-сервіс для планування та трекінгу задач», керівник проекту Люшенко Леся Анатолівна, ст. викладач, к.т.н., затверджені наказом по університету від «___»_____ 2020 р. №_____

2. Термін подання студентом проєкту «___» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - збір та опис вимог до веб-сервісу для планування та трекінгу задач;
   - аналіз мов програмування та технологій розроблення веб-сервісів;
   - розроблення веб-сервісу для планування та трекінгу задач;
   - аналіз розробленого веб-сервісу для планування та трекінгу задач.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма використання системи (креслення);
   - схема бази даних (креслення);
   - інтерфейс користувача (плакат);
   - авторська ілюстрація до сервісу(плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	10.11.2019	
2.	Розроблення та узгодження технічного завдання	27.11.2019	
3.	Розроблення структури веб-сервісу	14.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	27.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	02.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2020	
7.	Програмна реалізація веб-сервісу	12.03.2020	
8.	Тестування веб-сервісу	20.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	04.04.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	24.04.2020	
11.	Підготовка графічної частини дипломного проєкту	12.05.2020	
12.	Оформлення документації дипломного проєкту	24.05.2020	

Студент Уляна МАСЛЮК

Керівник проєкту Леся ЛЮШЕНКО

АНОТАЦІЯ

Дипломний проєкт присвячено розробці програмного забезпечення для планування та трекінгу задач.

В даному проєкті було розроблено архітектуру серверної та клієнтської частини веб-сервісу, структуру бази даних та створено дизайн інтерфейсів програмного забезпечення.

Веб-сервіс, що було розроблено, надає користувачам можливість створювати та зберігати завдання, надавати їм пріоритету, визначати кінцевий строк виконання, здійснювати збір та аналіз інформації про час, що був затрачений на виконання цих завдань. Також користувачі можуть переглянути візуалізовані дані про свою роботу, що полегшить їх аналіз своєї продуктивності. Завдання відображуються у вигляді дошки з картками. У системі передбачена можливість додавати до своєї дошки інших користувачів та ієрархічна структура всередині дошки, отже веб-сервісом може користуватися як одна людина, так і ціла команда.

ABSTRACT

This diploma project deals with development of software for planning and tracking tasks.

In this project, web service server and client part architecture and database structure were developed and software interfaces were designed.

The developed web service gives the ability to create and save tasks, set priority and deadlines, collect and analyze information about the time spent while working to its users. Users can also view visualized data about their work, which will facilitate their analysis of performance. Tasks are displayed in the form of a board with cards. The system provides the ability to add other users to your board and a hierarchical structure inside the board, so the web service can be used by one person or an entire team.

АННОТАЦИЯ

Дипломный проект посвящен разработке программного обеспечения для планирования и трекинга задач.

В данном проекте было разработано архитектуру серверной и клиентской частей веб-сервиса, структуру базы данных и создано дизайн интерфейсов программного обеспечения.

Разработанный веб-сервис предоставляет пользователям возможность создавать и сохранять задачи, выставлять им приоритеты, определять крайний срок выполнения, осуществлять сбор и анализ информации о времени, что было затраченное на выполнение этих задач. Также пользователи могут посмотреть визуализированные данные о своей работе, что облегчит их анализ своей производительности. Задачи отображаются в виде доски с карточками. В системе предусмотрена возможность добавлять к своей доске других пользователей и иерархическая структура внутри доски, следовательно веб-сервисом может пользоваться как один человек, так и целая команда.

ДП.045440-01-90 Веб-сервіс для планування та трекінгу задач. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-сервіс для	5	
	планування та трекінгу		
	задач. Технічне завдання		
ДП.045440-03-81	Веб-сервіс для	54	
	планування та трекінгу		
	задач. Пояснювальна		
	записка		
ДП.045440-04-51	Веб-сервіс для	4	
	планування та трекінгу		
	задач. Програма та		
	методика тестування		
ДП.045440-05-34	Веб-сервіс для	10	
	планування та трекінгу		
	задач. Керівництво		
	користувача		
ДП.045440-06-99	Веб-сервіс для	1	
	планування та трекінгу		
	задач. Діаграма		
	використання. Схема		
	алгоритму		
ДП.045440-07-99	Веб-сервіс для	1	
	планування та трекінгу		
	задач. База даних		
	системи. Схема даних		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-08-98	Веб-сервіс для	1	
	планування та трекінгу		
	задач. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

"___" _____ 2019 р.

ВЕБ-СЕРВІС ДЛЯ ПЛАНУВАННЯ ТА ТРЕКІНГУ ЗАДАЧ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Уляна МАСЛЮК

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-сервіс для планування та трекінгу задач.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розроблюване програмне забезпечення призначене для планування та трекінгу задач та часу, витраченого на їх виконання та подальшого аналізу даних про виконані задачі у вигляді графіків, побудованих за різними критеріями.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

- Таск-трекер має бути представленим у вигляді дошки, на яку можна додавати картки-задачі;
- задачам можна присвоїти виконавця, тип, надати пріоритет та встановити кінцевий строк виконання;
- трекінг часу;
- автоматичний підрахунок ресурсів, витрачених на виконання задач;
- візуалізація зібраних даних про роботу у вигляді графіків.

Розробку серверної частини виконати на платформі Node.js з використанням фреймворку Express, клієнтської – з використанням фреймворку Vue.js.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

- пояснювальна записка;
- програма та методика тестування;
- керівництво користувача;
- креслення:
  - «Діаграма використання системи»;
  - «Схема бази даних».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту……10.11.2019

Розроблення та узгодження технічного завдання……....27.11.2019

Розроблення структури веб-сервісу…….....14.12.2019

Підготовка матеріалів першого розділу проєкту…….27.12.2019

Розроблення дизайну сторінок та графічних елементів……..02.02.2020

Підготовка матеріалів другого розділу проєкту……...19.02.2020

Програмна реалізація веб-сервісу…….....12.03.2020

Тестування веб-сервісу...……....20.03.2020

Підготовка матеріалів третього розділу проєкту…….04.04.2020

Підготовка матеріалів четвертого розділу проєкту…….24.04.2020

Підготовка матеріалів графічної частини проєкту…….....12.05.2020

Оформлення технічної документації проєкту……..…24.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-СЕРВІС ДЛЯ ПЛАНУВАННЯ ТА ТРЕКІНГУ ЗАДАЧ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Уляна МАСЛЮК

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Таск-трекер (англ. task tracker – «відсліджувач завдань») – програмне забезпечення, що має функціональні можливості для створення, зберігання, перегляду та контролю завдань.

Фрилансер (англ. freelancer – «вільний митець») – вільнонайманець, який сам шукає собі проєкти, може одночасно працювати на декілька фірм. Фрилансер виконує роботу без укладання довгострокового договору з роботодавцем, найманий тільки для виконання певного переліку робіт (позаштатний працівник).

Тімлід (англ. team leader – «керівник команди») – це IT-фахівець, який керує своєю командою розробників, володіє технічною стороною, бере участь в роботі над архітектурою проєкту, займається рев'ю коду, а також розробкою деяких особливо складних завдань на проєкті.

CRM (англ. customer relationship management – «управління відносинами з клієнтами») – поняття, що охоплює концепції, котрі використовуються компаніями для управління взаємовідносинами зі споживачами, включаючи збір, зберігання й аналіз інформації про споживачів, постачальників, партнерів та інформації про взаємовідносини з ними.

Адаптивний інтерфейс – інтерфейс, що оптимально відображується користувачу незалежно від роздільної здатності та формату пристрою, з якого здійснюється перегляд сторінки.

SQL-ін'єкція – один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду.

XSS-атака (англ. Cross-Site Scripting – «міжсайтовий скриптинг») – тип атаки на веб-системи, що полягає у впровадженні у веб-системою

сторінку шкідливого коду (який буде виконаний на комп'ютері користувача при відкритті ним цієї сторінки) і взаємодії цього коду з веб-сервером зловмисника.

Callback hell (англ. пекло зворотніх викликів) – нечитаємий, складний код, що є багаторазово вкладеними блоками, що містять виклики на подальші вкладені блоки.

Фреймворк (англ. framework) – каркас, структура – заготовки, шаблони для програмної платформи, що визначають структуру системи програми. Також, це ПЗ, що облегшує та прискорює розробку та об’єднання різних модулей програмного проєкту.

СКБД (Система Керування Базами Даних) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

SQL – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних.

Веб-сокет – це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет.

Авторизація – надання певній особі або групі осіб прав на виконання певних дій; а також процес перевірки (підтвердження) даних прав при спробі виконання цих дій.

Аутентифікація – процедура встановлення належності користувачеві інформації в системі через перевірку пред'явленого ним ідентифікатора.

JS – JavaScript.

PDF (англ. Portable Document Format – «формат портативного документа») – відкритий формат файлу, створений і підтримуваний компанією Adobe Systems, для представлення двовимірних документів у незалежному від пристрою виведення та роздільної здатності вигляді.

ВСТУП

У сучасному світі, інформаційні технології є невід'ємною частиною повсякденного життя сучасної людини. Кожен користується набором програмного забезпечення, що спрощує його життя, наприклад, пошук необхідної інформації або прискорення робочих процесів.

Експеримент, що був проведений Ніколасом Блумом у 2010 році в китайській компанії «CTrip» довів, що віддалена робота підвищує продуктивність працівників на 13% [1]. Дійсно, в наш час існують самозайняті працівники (фрилансери), що працюють з дому та знаходять собі роботу через мережу Інтернет. Існують спеціальні онлайн-майданчики та соціальні мережі, що дозволяють їм просувати свої навички та спрощують пошук роботи, а сучасні технології роблять віддалену роботу доступною і більш продуктивною.

Специфіка роботи на фрилансі призводить до того, що нерідко фрилансери працюють над декількома проєктами одночасно. Отже, вони потребують програмне забезпечення, що зберігатиме завдання, що необхідно виконати, та їх строки виконання. Таке програмне забезпечення називається таск-трекером. Якщо до таск-трекеру додати функціональну можливість заміру часу, що був витрачений на виконання завдань, то отримане програмне забезпечення полегшить процес аналізу продуктивності та спростить спілкування з замовником щодо оплати виконаного завдання.

Таке програмне забезпечення потребують не лише фрилансери, що зазвичай працюють одні, а і цілі команди працівників, оскільки використання програмного застосунку такого типу значно підвищує продуктивність цілої команди та спрощує аналіз роботи кожного з учасників команди.

Дипломний проєкт присвячено розробці програмного забезпечення для планування та трекінгу задач.

Даний проєкт ставить за мету створення веб-сервісу, що поєднує в собі функції трекера задач та часу, витраченого на їх виконання, а також має функціональні можливості для аналізу даних про виконані задачі у вигляді графіків за різними критеріями.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА АКТУАЛЬНІСТЬ ТЕМИ

1.1. Актуальність систем автоматизації відслідковування статусу виконання задач та терміну їх виконання

В наш час кожен намагається бути максимально продуктивними, щоб працювати ефективніше для себе та оточуючих. Окрім бажання бути продуктивним, існує проблема того, що люди забувають про деякі задачі, які необхідно зробити, а це призводить до зниження продуктивності та погіршення стосунків з оточуючими: колегами, керівництвом або замовниками. Така тенденція породжує безліч програм для відслідковування статусу виконання задач та терміну їх виконання, або іншими словами таск-трекерів.

Існує безліч видів таск-трекерів, вони розрізняються за візуальною подачею завдань та функціональністю. Умовно таск-трекери поділяються на групи:

1. Списки завдань, які зручні для побутового використання, якщо необхідно занести список побутових справ, що необхідно виконати. Ці додатки мають обмежені функціональні можливості та дуже прості у використанні.
2. Візуальні дошки з картками, які використовують візуальний спосіб подачі інформації, що спрощує орієнтацію в великій кількості завдань. Такі таск-трекери мають більший попит серед розробників програмного забезпечення.
3. Комплексні професійні системи постановки та виконання завдань, які в своїй значній функціональності мають таск-трекери з контролем часу.

Перші дві групи мають недоліки:

- відсутність можливості контролювати час;
- відсутність обліку часу, який витрачено на виконання завдань;
- відсутність аналізу даних про статуси виконання завдань.

В свою чергу третя група використовується великими корпоративними користувачами та має надмірну функціональність, пов'язану з громіздкою системою управління та корпоративними регламентами.

Таким чином відсутній програмний додаток з таск-трекінгу для невеликих розробників програмного забезпечення, який би міг поєднати переваги перших двох груп та мав функціональні можливості з контролю й обліку часу виконання завдань, а також аналізу даних про статуси виконання завдань.

1.2. Аналіз існуючих програмних додатків для таск-трекінгу

1.2.1. Додаток «Harvest»

Додаток «Harvest» працює на для різних платформах: macOS, Windows, iOS та Android. Він підходить для фрилансерів та команд, які працюють віддалено. Додаток побудований у вигляді календаря на тиждень, де за кожен день відображується кількість часу, що була відпрацьована. Окрім функцій контролю термінів виконання та статусу виконання завдань, у додатку реалізовані деякі операції з обліку праці та формування рахунків. Є дуже зручна функція формування облікових звітів з термінів виконання завдань. Такі звіти можуть полегшити розрахунки з клієнтами, оскільки наглядно демонструють як витрачався час на їх замовлення. У додатку є графіки роботи, які надають значну кількість інформації, для керування роботою командою розробників: скільки часу працював член команди відносно тижневої норми, розподілення робочого часу за днями тижня та за видом завдання [2].

Переваги додатка:

- вбудована функція створення рахунків для PayPal Business Payments;

- можливість інтеграції зі програмами управління проєктами, створення рахунків, сервісами клієнтської підтримки, CRM;
- вбудована функція відстеження витрат, що включає в себе сканування чеків про оплату;
- терміни відстежуються програмами для десктопів і мобільних пристроїв.

Недоліки програмного додатка:

- складний інтерфейс, що підвищує складність роботи та відштовхує користувачів;
- відсутність елементів візуалізації;
- значні витрати для створення нових проєктів, що викликає незручності у фрилансерів;
- немає можливості редагування стандартних задач в проєкті;
- додаток є частково безкоштовним, він дозволяє користувачам створювати до двох безкоштовних проєктів.

1.2.2. Програмний комплекс «Jira Software»

Jira Software – це програмний продукт, створений спеціально, щоб керувати проєктами розробки програмного забезпечення.

Jira Software має функції створення завдань, призначення виконавців, установки пріоритетів і відстеження виконання цих завдань. Весь процес життєвого циклу додатки контролюється за допомогою Jira. Є інтеграція з іншими програмними продуктами компанії Atlassian.

Додаток має можливість створювати Scrum-дошки, що дозволяють agile-командам сфокусуватися на швидкому здійсненні ітерацій і введення нових змін. Можна створювати план дій, що буде візуалізуватися, та який можна підв’язати до робочих процесів на дошках. Є функція формування agile-звітів, тобто команди можуть отримати звіти з актуальною інформацією про те, як вони виконують спринт за спринтом [3].

Переваги програмного додатка:

- широка функціональність: можливість створювати задачі, надавати їм пріоритет, призначати виконавців, фіксувати час, витрачений на завдання тощо;
- інтерфейс, створений відповідно вимогам до створення інтерфейсів;
- існує безліч плагінів, що можна інтегрувати в додаток;
- велика кількість налаштувань під користувача, що робить роботу максимально комфортною.

Недоліки програмного додатка:

- велика кількість налаштувань призводить до того, що у нових користувачів виникає дуже багато проблем з цим процесом;
- неможливо швидко знайти необхідні інструменти через їх велику кількість;
- надмірна функціональність для невеликих команд розробників;
- програмний додаток є платним, а враховуючи, що ціна є статичною розрахунку на одного користувача в місяць, то рішення може бути занадто дорогим.

Програмний продукт рекомендовано використовувати для великих проєктів, які потребують значну кількість інструментів та налаштувань. Для середніх та невеликих проєктів це спричиняє незручності в управлінні розробки програмним продуктом.

1.2.3. Програмний продукт «Timely»

Цей програмний веб-додаток функціонує для платформ macOS, Windows, iOS, Android.

Timely містить стандартні функції таск-трекеру: створення задач, відстежування часу та статуси виконання. Є функціональні можливості для стеження за командою в режимі реального часу, тобто автоматичний часовий таск-трекінгу. Записується будь-яка активність, в залежності від

того, що робив користувачів. Програмний продукт підходить як для особистого використання, так і для роботи в команді.

Система має функцію самостійної адаптації під конкретного користувача, а саме: кожного разу коли користувач вводить дані, то система починає генерувати більш точні підказки [4].

Переваги програмного додатка:

- функція автоматичного часового таск-трекінгу;
- у керівництва є можливість перегляду того, що співробітники робили протягом робочого дня;
- вбудована панель Project Health Dashboard допомагає контролювати облікової робочий час, коли потрібно переконатися в тому, що команда не виходить за рамки встановленого бюджету.

Недоліки програмного додатка:

- складний для розуміння інтерфейс;
- функція автоматичного контролю за активністю співробітників може бути непотрібною, якщо розробка проводиться в офісі;
- відсутня безкоштовна версія програми.

1.2.4. Веб-застосунок «Trello»

Trello – веб-сервіс, що використовує канбан-дошки для управління проєктами з невеликою кількістю учасників.

Для організації завдань використовується канбан-дошка, тобто дошка з картками, які розподіляються за типами: заплановані, поточні та виконані задачі. У даного застосунку відсутні функціональні можливості для обліку часу, який витрачено на виконання завдань, та, як наслідок, аналіз даних про роботу.

Окрім веб версії застосунку, реалізовано рішення для платформ iOS та Android [5].

Переваги веб-застосунку:

- інтуїтивно зрозумілий інтерфейс;
- простий у використанні;
- безкоштовний.

Недоліки веб-застосунку:

- відсутня функціональна можливість контролю часу;
- відсутні інструменти для аналізу даних про роботу;

Проаналізуємо описані вище програмні рішення:

Таблиця 1

Порівняльна характеристика програмних додатків

Назва додатка	Harvest	Jira software	Timely	Trello
Інтерфейс	Складний	Складний	Складний	Простий
Аналіз даних про статуси виконання завдань	Є	Є	Є	Відсутній
Контроль за терміном виконання	Відсутній	Є	Відсутній	Є
Облік часу	Є	Є	Є	Відсутній
Масштаб проєкту	середні та великі ІТ компанії	Великі ІТ компанії	Малі та середні ІТ компанії	Малі та середні ІТ компанії
Наявність безкоштовної версії	Є	Відсутня	Відсутня	Є

Більшість додатків має складний для розуміння інтерфейс. Три з чотирьох розглянутих програмних застосунків мають функції обліку часу та

аналізу статусів виконаних завдань, проте лише половина має контроль за терміном виконання. Майже всі додатки не мають безкоштовної версії або мають дуже обмежені функціональні можливості в безкоштовній версії.

Проаналізувавши дані з таблиці, робимо висновок що не існує програмного застосунку, що поєднував би в собі всі необхідні функціональні можливості, мав простий інтерфейс та був безкоштовним.

1.3. Актуальність розробки застосунку

Як вже було зазначено вище, існує проблема відсутності програмного забезпечення систем автоматизації відслідковування статусу виконання задач та терміну їх виконання для невеликих компаній та команд розробників програмного забезпечення.

Це програмне забезпечення має бути реалізовано в формі веб-застосунку, що поєднує в собі зручну візуалізацію роботи з завданнями з функціями контролю їх виконання та часу. В цьому додатку можна створити дошку задач як для команди, так і для окремого користувача та додати функціональні можливості, що візуально представлятиме дані про виконану роботу. Інформація може подаватися у вигляді звітів з графіками по роботі кожного з члена команди. Графіки можуть бути створені по різним критеріям, наприклад: який відсоток часу яким типом задач займався даний член команди або навпаки, який член команди виконав більше завдань певного типу. Такий інструмент буде корисним і для окремого користувача, що не має команди. Переваги такої візуалізації у тому, що наглядно демонструється різноманітні дані про роботу працівника, що облегшує прийняття управлінських рішень.

Окрім того веб-додаток повинен реалізовувати:

- фіксацію даних про виконані завдання співробітниками;
- функцію синхронізації між задачами та витраченим часом на них, що дозволить приймати коректні рішень щодо їх матеріальної мотивації;

- функцію автоматичного підрахунку часу та видатків на оплату;
- функцію автоматичного аналізу даних про статуси виконання завдань та повноту і якість виконання;
- зручний та простий в використанні інтерфейс з максимальною візуалізацією.

1.4. Вимоги до веб-сервісу для планування та трекінгу задач

Визначимо основні вимоги до веб-застосунку, а саме: вимоги до інтерфейсу апаратні вимоги, вимоги до функціональності, операційні вимоги та вимоги до безпеки.

1.4.1. Вимоги до інтерфейсу

Інтерфейс має забезпечувати:

- сумісність з можливостями та потребами користувача;
- простоту переходу між виконанням різних функцій;
- має бути user-friendly;
- задачі мають бути представлені у вигляді дошок з картками.

1.4.2. Апаратні вимоги

Комп’ютер або смартфон з браузерами:

- Google Chrome (від версії 50.0);
- Firefox (від версії 45);
- Internet Explorer (від версії 9);
- Safari (від версії 9);
- Opera (від версії 40);
- необхідний доступ в мережу Інтернет.

1.4.3. Вимоги до функціональності

- таск-трекер має бути представленим у вигляді дошки, на яку можна додавати картки-задачі;

- задачам можна присвоїти виконавця, тип, надати пріоритет та встановити кінцевий строк виконання;
- трекінг часу;
- автоматичний підрахунок ресурсів, витрачених на виконання задач;
- візуалізація зібраних даних про роботу у вигляді графіків за критеріями:
    - динаміка кількості робочих годин за тиждень, місяць;
    - скільки годин на який тип задач було витрачено;
    - відсоток завдань, що були закінчені після кінцевого строку здачі;
    - порівняння робочих годин кожного працівника за спеціальністю.

1.4.4. Операційні вимоги

Веб-застосунок повинен забезпечувати:

- час відгуку не має перевищувати 1 секунду;
- використання бази даних для збереження даних користувача;
- сервіс має працювати у нормальному режимі при тисячі одночасно активних сеансах;
- сервіс має бути доступним 24 години на добу.

1.4.5. Вимоги до безпеки

Веб-застосунок має забезпечувати:

- шифрування паролів користувачів;
- захист від ін’єкцій до бази даних;
- захист від JavaScript-ін’єкцій;
- перевірку на валідність введених даних користувачем.

1.5. Висновки до розділу

Отже, створення програмного забезпечення для планування та трекінгу задач набуває все більшої актуальності на фоні пришвидшення темпу життя людини, в умовах постійного росту кількості поточних завдань та масштабних задач. Часто, працюючи на конкретну мету, людина забуває про додаткові завдання, прохання близьких та справи, відкладені на «пізніше». Щоб охопити усю різноманітність повсякденної діяльності людини, врахувавши моменти пріоритетності та дедлайнів, було вирішено розробити додаток з можливістю менеджменту робочих задач, побутових справ та можливістю контролювати час виконання кожної із задач. Таким чином актуальним є розробка програмного забезпечення з функціональними можливостями декількох існуючих типів таск-трекерів і створення універсального додатку з можливістю фіксації даних про виконання задач, контролем витраченого часу на їх виконання, та можливістю планувати фінансову мотивацію, автоматичного контролю за якістю виконання завдань і простим інтерфейсом з максимальною візуалізацією.

Деталізувавши вимоги до веб-додатку, формуємо вимоги до інтерфейсу, апаратні, функціональні, операційні вимоги та вимоги до безпеки додатку:

- веб-додаток повинен працювати усіма популярними серед користувачів браузерами, пристрій повинен мати доступ до мережі Інтернет;
- необхідно забезпечити простий, user-friendly інтерфейс, візуалізацію задач у вигляді дошок з картками;
- забезпечити наступні функціональні можливості: присвоєння задачам типи, виконавців, терміни та трекінг часу на виконання, автоматизувати підрахунки ресурсів на виконання задач та аналіз ефективності їх виконання;
- забезпечити час відгуку не більше секунди, базу даних для

збереження даних користувача, безперебійну, цілодобову роботу сервісу за умов високого трафіку;

- створити умови для безпечного використання додатку: шифрування персональних даних, захист від ін’єкцій бази даних та JavaScript-ін’єкцій, перевірку на валідність введених даних користувачем.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Обґрунтування вибору мови програмування

Оскільки ПЗ, що розроблюється, є веб-сервісом, то мова програмування буде обрана зі списку найпопулярніших мов, на яких розроблюються веб-застосунки. Окремо будуть обрані мови для серверної та клієнтської частин.

2.1.1. Вибір мови програмування серверної частини

Мова програмування C#

C# – це об'єктно-орієнтована мова програмування, що підтримує імперативну, структурну, логічну, функціональну та декларативну парадигми. Мова була фірмою розроблена компанією Microsoft. Має сувору статичну типізацію. Підтримує механізми об’єктно-орієнтованого програмування: успадкування, інкапсуляція та поліморфізм [6].

Для цієї мови програмування існує велика кількість бібліотек та фреймворків. Найвідоміший фреймворк, що існує це Microsoft .NET — крос-платформна технологія, що дозволяє створювати віконні (Windows Forms або WPF) та веб (ASP.NET) застосунки [7].

Переваги:

- велика кількість шаблонів та бібліотек;
- через сувору типізацію код є зрозумілим і простим;
- використання парадигми об’єктно-орієнтовного програмування.

Недоліки:

- для роботи з даною мовою необхідно завантажувати велику кількість інструментів, що займають багато пам’яті;
- наявність безкоштовної ліцензії лише для ОС Windows.

Платформа Node.JS

Node.js – програмна платформа, що перетворює JavaScript, що застосовується як вбудована мова для програмного доступу до об'єктів додатків в мову загального призначення. Має динамічну типізацію. Платформа реалізує подійно-орієнтовану парадигму [8].

Платформа має велику кількість фреймворків, що спрощують створення серверної частини, наприклад Express [9].

Express – це мінімалістичний і гнучкий веб-фреймворк для Node.js, що надає великий набір функцій для створення мобільних і веб-додатків [10].

Переваги:

- висока швидкодія через використання асинхронної моделі запуску коду;
- можливість застосовувати одну мову як на серверній частині, так і на клієнтській;
- є велика кількість сторонніх бібліотек та інструментів, що пришвидшує швидкість розробки проєкту;
- є власний встановлювач пакетів npm, що спрощує роботу з пошуку та встановлення сторонніх бібліотек на проєкт.

Недоліки:

- неправильна реалізація подій може призвести до callback hell;
- необхідна висока кваліфікація розробника для ефективної роботи з парадигмою мови.

Мова програмування Ruby

Ruby – високорівнева об'єктно-орієнтована мова, що реалізує парадигму об’єктно-орієнтовного програмування. Має динамічну типізацію [11].

Для цієї мови створено об'єктно-орієнтований фреймворк Ruby on Rails, що реалізує архітектурний шаблон Model-View-Controller який застосовується для розробки веб-застосунків [12].

Переваги:

- висока швидкість розробки проєктів за рахунок того що існує велика кількість готових рішень;
- високий рівень захисту даних, оскільки сама мова виключає можливість SQL ін'єкції і XSS атаки;
- є велика кількість готових бібліотек та фреймворків.

Недоліки:

- низька швидкодія;
- недостатня кількість прикладів та документації, що робить мову важкою у вивченні.

Мова програмування Python

Python – високорівнева мова програмування загального призначення. Стандартна бібліотека включає великий обсяг корисних функцій. Python підтримує такі парадигми: структурну, об'єктно-орієнтовану, функціональну та імперативну.

Має динамічну типізацію, автоматичне керування пам'яттю та підтримує багатопоточність [13].

Переваги:

- легко читати та редагувати код;
- наявна велика кількість сторонніх бібліотек, що пришвидшує розробку проєкту.

Недоліки:

- низька швидкодія;
- неможливість модифікувати вбудовані класи.

Обрана мова програмування

Переваги аналізованих мов, є схожими: велика кількість сторонніх бібліотек та зрозумілий код. Проте, недоліки відрізнялись, що і вплинуло на вибір мови програмування серверної частини:

- оскільки веб-застосунок буде розроблюватися на операційній системі Linux, то критичним є те, що для створення проєкту на

мові програмування C# необхідно отримати ліцензоване програмне забезпечення, яке є платним для даної операційної системи;

- основним недоліком мови Ruby є недостатня кількість документації та прикладів, що робить розробку проєкту довшою та складнішою;
- мова Python має низьку швидкодію, що є важливим в розроблюваному проєкті, тож, як наслідок, було прийнято рішення не використовувати її.

Проаналізувавши та оцінивши найпопулярніші мови програмування для розробки веб-застосунків, було зроблено вибір у сторону платформи Node.js. Обрана мова швидко розвивається та має велику кількість сторонніх бібліотек та платформ, наприклад Express, що пришвидшує та робить більш комфортною розробку веб-серверів.

2.1.2. Вибір фреймворку для програмування клієнтської частини

React

React – це декларативний, ефективний і гнучкий JavaScript фреймворк для створення користувацьких інтерфейсів. Він дозволяє збирати складний UI з маленьких ізольованих шматочків коду, званих «компонентами» [14].

Переваги:

- використовує віртуальний DOM;
- додатки, створені на React, витримують великі навантаження;
- забезпечує просту міграцію між версіями.

Недоліки:

- невпорядкованість документації робить опанування фреймворку важким;
- для вивчення всіх можливостей фреймворку потрібен тривалий час.

Angular

Angular – платформа для розробки клієнтської частини веб-застосунків, написана на мові TypeScript командою розробників з компанії Google [15].

Переваги:

- велика кількість різноманітних функцій та інструментів;
- використання MVVM (Model-View-ViewModel) шаблону, що дозволяє розробникам окремо працювати в одному розділі програми з використанням одного і того ж набору даних;
- двостороння прив'язка даних мінімізує ризики можливих помилок.

Недоліки:

- складний синтаксис;
- при переході від старої версії до нової можуть виникнути проблеми з міграцією.

Vue.js

Як і React та Angular, Vue.js – JavaScript-фреймворк, призначений для створення користувацьких інтерфейсів на основі моделей даних. Замість того, щоб оновити дані користувачів тільки у DOM, модель даних буде синхронізована для відображення змін. Vue працює за допомогою реактивного зв’язування даних, тобто при оновленні, наприклад, поля форми, основна модель даних оновиться автоматично. Якщо інші частини веб-сайту, створеного з використанням фреймворку Vue.js також прив’язані до тієї самої моделі даних, їх вміст також оновиться [16].

Переваги:

- має простий та зрозумілий синтаксис;
- небагато важить, що підвищує швидкість завантаження сторінок;
- висока швидкість розробки завдяки можливості використання будь-яких шаблонів та доступності документації.

Недоліки:

- мало інструментів для відлагодження проєкту;
- відсутність бібліотек та плагінів.

Обраний фреймворк для програмування клієнтської частини

Проаналізувавши найпопулярніші фреймворки для розробки клієнтської частини, було виявлено суттєві недоліки:

- невпорядкованість документації фреймворку React підвищує час розробки проєкту, що є критичним;
- проблеми з міграцією з старої версії до новішої в Angular можуть загальмувати розвиток проєкту при подальшій розробці.

Врахувавши наведені вище переваги та недоліки, було обрано Vue.js, оскільки реалізація проєкту з використанням даного фреймворку потребує мінімальних затрат часу порівняно з іншими, що були розглянуті. Також, обраний фреймворк має високу швидкодію, що є дуже важливим в розроблюваному проєкті.

2.2. Обґрунтування вибору системи керування базами даних

MySQL

MySQL – одна з найпопулярніших [17] реляційна система управління базами даних. Підходить для малих та середніх додатків. Це безкоштовний пакет програм, що активно розвивається: розширюються функціональні можливості та підвищується рівень безпеки.

Можливість створювати різні типи таблиць, та застосовувати різні інструменти щоб виконувати обробку даних, що зберігаються в них, роблять СКБД гнучкою [18].

Переваги:

- є велика кількість плагінів що спрощують роботу;
- широкий набір стандартних інструментів для роботи з даними; висока швидкодія системи.

Недоліки:

- великі затрати часу на розробку через відсутність автоматичного виконання функцій, наприклад, створення резервних копій;
- для безкоштовної версії доступна тільки платна підтримка.

Redis

Redis – резидентна система керування базами даних класу NoSQL, що працює зі структурами даних типу «ключ - значення», що зберігає дані в пам'яті, доступ до яких здійснюється по ключу доступу.

Дана СКБД зберігає базу даних в оперативній пам'яті та має механізми для забезпечення постійного зберігання даних на фізичних накопичувачах. Всі дані зберігаються у вигляді словника, в якому ключі пов'язані зі своїми значеннями. Особливістю Redis є те, що значення цих ключів не обмежуються лише даними типу «рядок». Підтримуються такі абстрактні типи даних: рядки, списки, множини, хеш-таблиці та впорядковані множини [19].

Переваги:

- висока швидкодія;
- прості та зручні формати даних;
- є вбудовані алгоритми що видаляють застарілі дані з бази даних.

Недоліки:

- розмір бази даних обмежений пам'яттю що доступна;
- зниження швидкодії при масштабуванні бази даних;
- відсутній вбудований контроль доступу, що знижує безпеку даних;
- має специфічну область застосувань: для зберігання токенів авторизації, як кеш, брокер повідомлень, тощо.

MongoDB

MongoDB – це документо-орієнтована база даних, що заснована на принципі зберіганні документів в BSON (Binary JSON) форматі. Кожен

запис це документ без жорстко заданої схеми, який може містити інші вкладені документи [20].

Переваги:

- гнучка мова для формування запитів;
- можливість швидко оновлювати базу даних;
- ефективне зберігання двійкових даних великих обсягів, таких як фото і відео;
- автоматичний запис та збереження операцій, що модифікують дані в базі;
- вбудовані інструменти для масштабування бази даних: асинхронна реплікація та шардинг;
- забезпечує розподілений доступ до даних.

Недоліки:

- неможливо часткове оновлення документа, оскільки атомарна одиниця є цілим документом;
- відсутній інструмент для вирішення конфліктів між деагрегованими даними, що може знизити швидкість розробки.

Обрана система керування базами даних

Розглянувши та проаналізувавши СКБД, було виявлено такі критичні недоліки:

- при роботі з системою MySQL можуть виникнути додаткові затрати часу на розробку через відсутність автоматичного виконання функцій, які виконуються автоматично в інших СКБД, що є критичним;
- розроблюваний веб-застосунок не входить до обмеженої області застосування СКБД Redis, що робить використання даної системи недоцільним.

Для програмного додатку, що розроблюється, було обрано СКБД MongoDB, оскільки дана система є гнучкою та має широкий набір інструментів для масштабування, що є важливим, оскільки розроблюваний

веб-застосунок буде швидко наповнювати базу даними. Також дана СКБД є простою у використанні та має високу швидкодію. Дана СКБД є сумісною із Express, що робить роботу з БД зручною.

2.3. Висновки до розділу

У даному розділі було розглянуто інструменти технологій розробки: мови програмування серверної частини, фреймворки для програмування клієнтської частини та системи керування базами даних, з огляду на розроблюваний веб-застосунок. Було детально проаналізовано недоліки та переваги таких мов програмування серверної частини, як C#, Ruby, Python та Node.js, фреймворків React, Angular та Vue.js та СКБД MySQL, Redis та MongoDB.

Було обрано платформу для програмування серверної частини Node.js, оскільки вона є зручною в роботі, має зрозумілий синтаксис та швидко розвивається. Має велику кількість бібліотек та фреймворків, що пришвидшують та полегшують розробку.

Для створення клієнтської частини було обрано фреймворк Vue.js оскільки він об’єднує в собі особливості інших розглянутих фреймворків, що робить розборку швидкою та мінімізує можливість виникнення помилок. Також, інтерфейси, розроблені з використанням даного фреймворка, мають більшу швидкодію, ніж інші, оскільки він є найлегшим.

Серед розглянутих СКБД було обрано MongoDB, оскільки дана система має високу швидкодію та є простою у використанні. Також, вона має широкі функціональні можливості, серед яких є інструменти для масштабування та роботи з різними типами дати та часу, що є важливим в розроблюваному веб-застосунку.

3. РОЗРОБЛЕННЯ ВЕБ-СЕРВІСУ

3.1. Загальний опис програми

Веб-сервіс було розроблено з урахуванням всіх вимог до інтерфейсу, безпеки та функціональним можливостям, що були описані у підрозділі 1.4. Для забезпечення ієрархічної структури всередині проєкту, у системі передбачено дві ролі: «Власник» та «Користувач».

Власник

Проводить усі початкові дії, що стосуються створення нової дошки та отримує автоматично згенеровані ідентифікатор та пароль.

Має можливість створювати, редагувати та видаляти списки з картками, а також створювати нові картки с завданнями для будь-якого члена команди, видаляти будь-які та редагувати вже створені ним (змінювати назву, кінцевий термін виконання, пріоритет, тип завдання).

Має доступ до даних про час та роботу кожного члена команди.

Користувач

Може приєднатися до дошки за допомогою ідентифікатору та паролю.

Має можливість створювати картки з завданнями для будь-якого члена команди з роллю «Користувач» і видаляти та редагувати (змінювати назву, кінцевий термін виконання, пріоритет, тип завдання) завдання, що були створені ним.

Може переглядати свої дані про час та роботу.

Кожен користувач редагувати інформацію про себе на власній сторінці: змінювати ім’я, прізвище, фото, пароль та електронну пошту.

На початку роботи з веб-сервісом користувач потрапляє на головну сторінку де має можливість ввести дані для авторизації у системі (адрес електронної пошти чи юзернейм та пароль), авторизуватися з використанням сервісу Google або перейти на сторінку реєстрації, щоб створити аккаунт.

Для створення аккаунту на сторінці реєстрації, користувач має ввести про себе дані: адрес електронної пошти, пароль, ім’я, прізвище та юзернейм.

Після вдалої авторизації, користувач потрапляє на сторінку «Мої дошки», на якій відображуються всі дошки, учасником яких є даний користувач. Дошкам можна надати пріоритет із зірочкою, тоді вони відображуватимуться на сторінці вище за дошки без зірочки. З цієї сторінки користувач має можливість перейти на обрану дошку або на сторінки з налаштуваннями, інформацією про свій акаунт та даними про витрачений час на виконання завдань.

На сторінці обраної дошки користувач має змогу переглядати членів команди та їх ролі, всі існуючі завдання та їх виконавців та запускати відлік часу для завдань, які має виконати. Також, якщо користувач має роль «Власник» то він може переглядати затрачений іншим членом команди час на виконання обраного завдання. При натисканні на фото члена команди, відкриється модальне вікно з даними про його аккаунт. Користувач може переглянути завершені завдання, для цього він має натиснути на кнопку з надписом «Завершені завдання». Після цього з’явиться модальне вікно, що містить в собі список виконаних завдань, які можна сортувати за виконавцем та часовим проміжком.

На сторінці свого аккаунту, користувач може переглядати введену раніше інформацію про себе.

На сторінці з даними про час, затрачений на виконання завдань, користувач має можливість переглядати дані за обраний ним проміжок часу за типом задач чи за проєктами.

При переході на сторінку налаштувань, користувач має можливість змінити дані про себе та вийти із мережі.

3.2. Аналіз функціональних вимог до програмного додатку

При розробленні веб-сервісу кожній з функціональних вимог було надано пріоритет: значення «1» надане ключовим функціональним можливостям. Розподіл пріоритетів наведено у таблиці 2.

Таблиця 2

Функціональні вимоги до веб-сервісу для трекінгу задач та часу, затрачених на них

Код вимоги	Назва вимоги	Пріоритет вимоги
01	Реєстрація в системі	1
02	Авторизація в системі за допомогою логіну або адреси електронної пошти та паролю	1
03	Авторизація в системі за допомогою сервісу Google	2
04	Можливість створити дошку	1
05	Можливість додавати інших користувачів до створеної дошки	2
06	Можливість виділяти зірочкою обрану дошку	3
07	Можливість створити колонку з завданнями	2
08	Можливість створити завдання на дошці	1
09	Можливість визначати тип завдання	2
10	Можливість визначати кінцевий строк виконання завдання	2
11	Можливість засікання часу на виконання конкретного завдання	1

Продовження табл. 2

12	Можливість переглядати дані про час, затрачений на виконання конкретного завдання	2
13	Можливість переглядати дані про час, затрачений на виконання завдань за певний проміжок часу	2
14	Можливість переглядати затрачений час на виконання задач певного типу	1
15	Автоматичний підрахунок ресурсів, витрачених на виконання задач	2

3.3. Архітектура системи

Оскільки розроблюване програмне забезпечення є веб-сервісом, то було вирішено побудувати клієнт-серверну архітектуру. Вона має розподілену структуру додатків, що розбиває запити між постачальниками ресурсу, що називаються серверами, і запитувачами послуг, що називаються клієнтами.

В переважній більшості веб-проєктів, дана архітектура реалізується за допомогою шаблону Model-View-Controller (MVC) . При використанні даного шаблону, користувацький інтерфейс програми розділяється на три окремі частини: Model (описує бізнес-логіку, правила валідації та формат даних, що використовуються у веб-застосунку), View (визначає як відображати дані, отримані від Model sкористувачу), Controller (в залежності від користувацького запиту розподіляє команди між View та Model, здійснює фільтрацію отриманих даних) [21].

Оскільки для розробки користувацького інтерфейсу веб-сервісу було обрано Vue.js, то було застосовано архітектурний шаблон Model-View-ViewModel (MVVM). В даній архітектурі, Model та View виконують такі ж самі ролі як і в MVC, проте замість Controller є ViewModel – він містить в

собі команди, якими Model керує View, а також сам Model, що був перетворений до View [22].

Для створення клієнт-серверної архітектури, було розроблено 3 модулі:

- веб-сервер;
- веб-клієнт;
- модуль для роботи з базою даних.

Діаграму, на якій зображено взаємодію між модулями системи та користувачем, можна побачити на рис. 1.

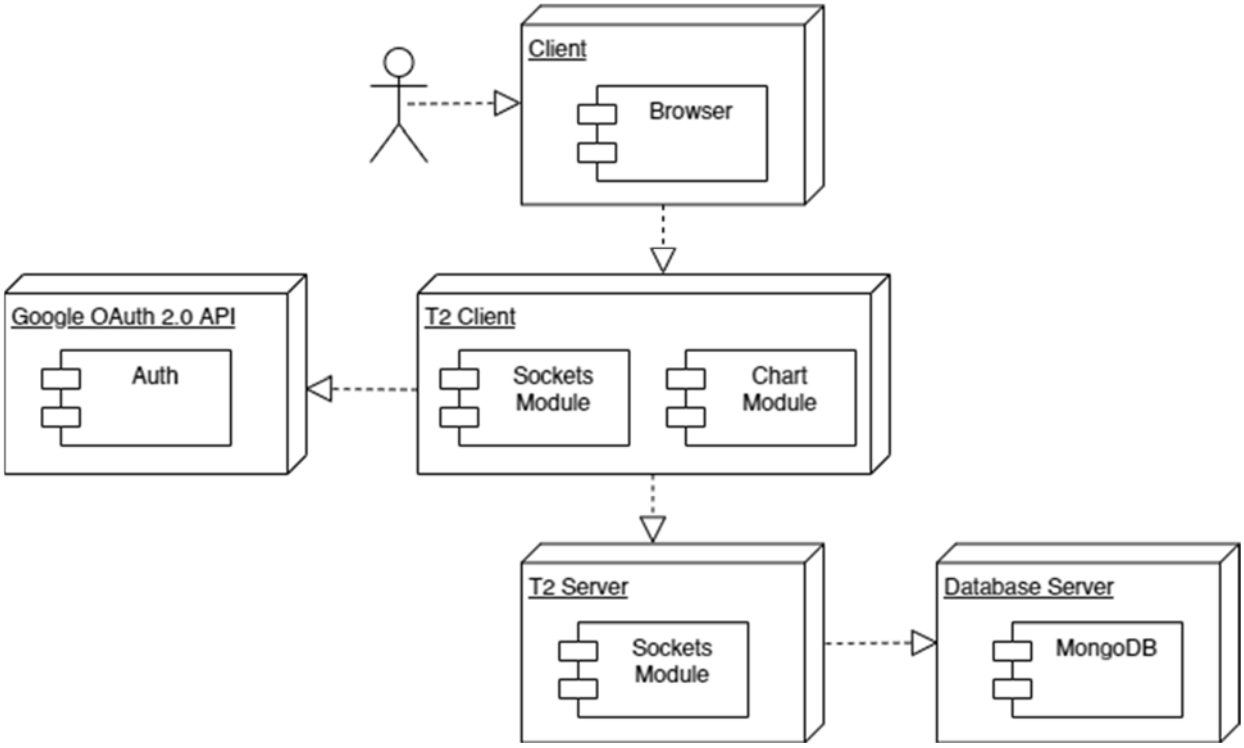


Рис. 1. Діаграма модулів системи

Веб-сервер

Веб-сервер оброблює запити, що надходять від користувача через веб-клієнт та відповідає на них, взаємодіє із сервером бази даних: заносить нові дані до таблиць бази та запитує, опрацьовує або видаляє вже існуючі. Також, шифрує пароль при реєстрації, проводить аутентифікацію та подальшу

авторизацію з використанням двох стратегій: локальної та за допомогою Google OAuth 2.0 API.

Було розроблено модуль для взаємодії з веб-клієнтом через сокети – Sockets Module.

Веб-клієнт

Надає користувачу графічний інтерфейс для використання функціональних можливостей веб-застосунку. Було розроблено два модулі:

- Sockets Module для взаємодії з веб-сервером через сокети;
- Chart Module для візуалізації даних про час, що був затрачений на роботу користувача.

Взаємодія з СКБД

Для створення веб-сервісу, було розроблено 7 моделей, з яких 3 головні, що зберігають дані про основні сутності у веб-сервісі :

- User;
- Board;
- Task,

та 4 допоміжні:

- ExecutionTime;
- userBoard;
- BoardUser;
- Column,

що є частинами основних сутностей.

User

У цій моделі містяться дані про користувачів. Для даної таблиці розроблено допоміжну модель – userBoard. Детальніша інформація про назви полів моделей та їх типи наведені у таблицях 3 та 4.

Таблиця 3

Поля моделі User

Назва поля	Тип поля
id	Number AUTOINCREMENT
email	String
username	String
password	String
firstName	String
lastName	String
city	String
birthdayDate	Date
photo	Binary
boards	userBoard[]

Таблиця 4

Поля моделі userBoard

Назва поля	Тип поля
boardId	Number
isFavourite	Boolean

Board

Дана модель зберігає дані про дошки, задачі, що містяться в них та членів команди та їх ролі. Для реалізації моделі було створено дві допоміжні: BoardUser та Column. Більш докладна інформація про назви полів моделей Board, BoardUser, Column та їх типи наведені у таблицях 5, 6, 7.

Таблиця 5

Поля моделі Board

Назва поля	Тип поля
id	Number AUTOINCREMENT
name	String
users	BoardUser[]
date	Date
columns	Column[]

Таблиця 6

Поля моделі boardUser

Назва поля	Тип поля
userId	Number
role	String
salary	Number

Таблиця 7

Поля моделі Column

Назва поля	Тип поля
id	Number
name	String
tasks	Task[]

Task

Модель Task зберігає дані про завдання. В ній міститься допоміжна модель executionTime. Детальніша інформація про назви полів моделей та їх типи наведена у таблицях 8 та 9.

Таблиця 8

Поля моделі Task

Назва поля	Тип поля
id	Number AUTOINCREMENT
name	String
description	String
priority	Number
executionTime	executionTime[]
createdAt	Date
type	String
deadline	Date
deadlineIsFailed	Boolean
executorId	Number

Таблиця 9

Поля моделі executionTime

Назва поля	Тип поля
start	Date
end	Date

Схему архітектури розробленої бази даних можна переглянути на рис. 2.

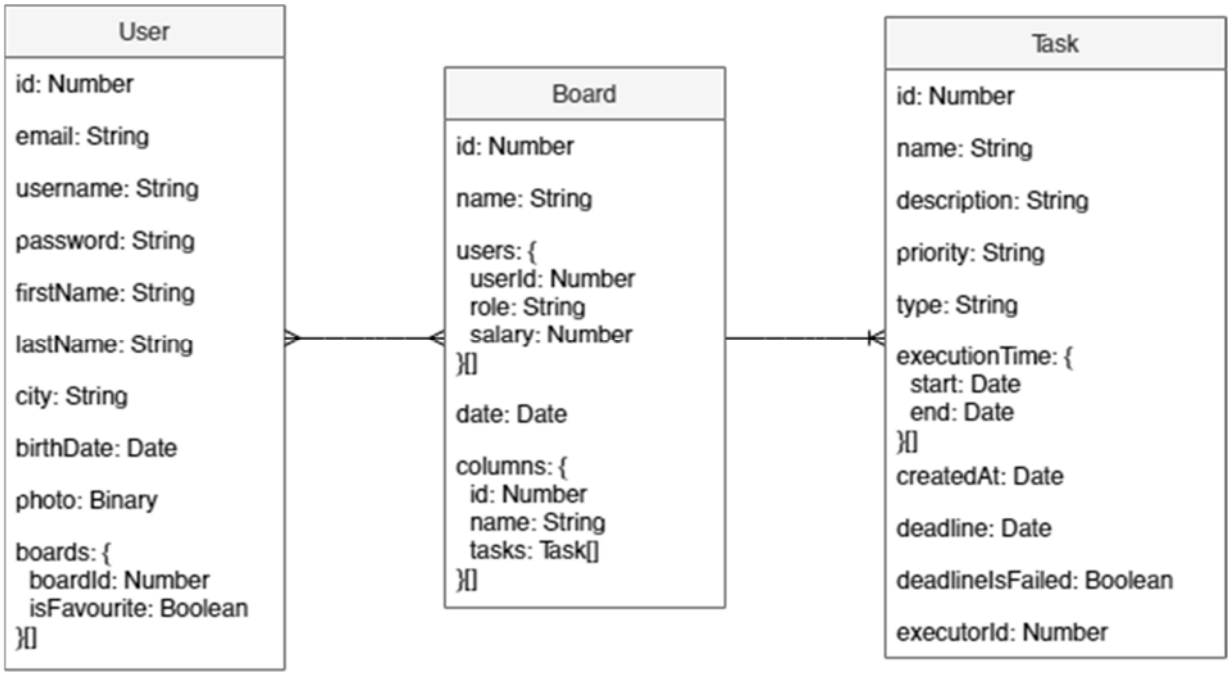


Рис. 2. Схема бази даних системи

3.4. Особливості реалізації

Головною архітектурною особливістю є використання шаблону проєктування Model-View-ViewModel, замість найбільш розповсюдженого Model-View-Controller, оскільки розробка ведеться з використанням фреймворку Vue.js.

Даний архітектурний шаблон має переваги:

- можливість спільної роботи декількох членів команди над одним і тим ж інтерфейсом. Оскільки користувацький інтерфейс моделі та відповідний код розділені, то двоє розробників можуть одночасно працювати над однією і тою самою моделлю без потреби спільно працювати в одному і тому ж файлі;
- повторне використання коду. Наявність одного і того ж коду в різних місцях підвищує складність підтримки коду та спричинює проблеми при масштабуванні проєкту. Оскільки даний шаблон є роздільним, то він дозволяє використовувати один шар логіки повторно в різних моделях проєкту;
- простота в тестуванні. Оскільки в даному шаблоні

користувацький інтерфейс розділений з логікою програми, то тестування проводиться програмно, а не через користувацький інтерфейс, що, пришвидшує та полегшує процес тестування.

Шаблон проєктування складається з трьох компонентів:

- Model;
- View;
- ViewModel.

Model

В даному модулі описана логіка роботи з даними. Було створено схеми, що описують основні моделі у базах даних.

User – модель, що має:

- унікальний ідентифікатор;
- адресу електронної пошти користувача;
- користувацький псевдонім;
- ім’я;
- прізвище;
- місто;
- дата народження;
- аватар;
- посилання на сутності моделі Board.

Board – модель, що має:

- унікальний ідентифікатор;
- пароль;
- назву;
- список користувачів, що мають доступ до дошки їх ролі та тарифну ставку;
- дату створення;
- колонки з посиланнями на сутності моделі Task, що містяться у цих колонках.

Task – модель, що має:

- унікальний ідентифікатор;
- назву;
- опис;
- пріоритет;
- тип;
- часові проміжки, які виконувалось завдання;
- дату створення;
- кінцевий строк здачі;
- мітку виконання після кінцевого строку здачі;
- посилання на сутність User, якій назначене дане завдання.

View

Даний модуль містить графічний інтерфейс. Також, він є споживачем подій, що надаються ViewModel.

Було розроблено 7 графічних інтерфейсів:

- сторінка для входу у систему;
- сторінка реєстрації у системі;
- сторінка з інформацією про користувача;
- сторінка зі списком дошок;
- сторінка дошки з колонками та завданнями в них;
- сторінка з візуалізованими даними про роботу;
- сторінка з налаштуваннями.

ViewModel

Містить набір команд, якими може користуватися View для того щоб змінювати Model, а також Model, що був перетворений до View.

Для візуалізації даних про роботу користувача було створено модуль Chart, що розроблено з використанням фреймворку ApexCharts.js.

ApexCharts.js - це бібліотека, що спрощує створення інтерактивних графіків та діаграм [23].

Модуль будує такі графіки:

- гістограма, що показує кількість робочого часу в день за тиждень, місяць, рік;
- гістограма, що показує кількість виконаних завдань за тиждень, місяць, рік;
- кругова діаграма, що показує відношення кількості затраченого часу, на виконання завдань відносно їх типу;
- гістограма, що показує кількість зароблених грошей по тижням, місяцям, рокам;
- кругова діаграма, що показує відношення кількості виконаних завдань відносно їх типу;
- кругова діаграма, що показує відношення завдань що були закінчені після кінцевого строку здачі до всіх виконаних;
- гістограма, що відображає кількість робочих годин кожного працівника за спеціальністю;
- гістограма, що відображає кількість виконаних завдань одного типу працівниками однієї спеціальності.

3.5. Висновки до розділу

Даний розділ присвячено загальній характеристиці розроблюваного веб-сервісу. Було враховано усі вимоги, поставлені до інтерфейсу, безпеки та функціоналу програми, створено ієрархічну структуру в середині проєкту, яка відкриває різні можливості програми користувачам з різнем рівнем відповідальності (відповідно до ієрархії працівників у компанії).

Опис архітектури системи обґрунтовує вибір клієнт-серверної архітектури програми, спосіб реалізації за допомогою шаблону MVC. Проведено порівняння обраного архітектурного шаблону MVVM з найпопулярнішим. MVC. Головною особливістю проєкту є використання шаблону проєктування Model-View-ViewModel. Описано переваги

архітектурного шаблону та кожен з його компонентів: Model, View та ViewModel.

У розділі детально описано роботу різних частин веб-сервісу, зокрема модуля Chart, розробленого з використанням фреймворку ApexCharts.js. Модуль допомагає користувачам отримати дані про кількість робочого часу за різний період, кількісні показники виконаних завдань, робочих годин кожного

з працівників, дані про співвідношення кількості завдань до їх типу та інші дані, які будуть запитані користувачами. Статистику модуль подає у вигляді гістограм та діаграм.

4. АНАЛІЗ РОЗРОБЛЕНОГО ВЕБ-СЕРВІСУ

4.1. Аналіз реалізованого веб-сервісу

У результаті розробки програмного забезпечення було створено веб-сервіс, що має функціональні можливості для ефективного тайм-менеджменту як однієї людини, так і цілої команди, а саме створення дошок, на яких можна розміщувати завдання, назначати їм виконавців, кінцевий термін виконання та пріоритет, а виконавець може заміряти час, що був витрачений на виконання. Сервіс оброблює зібрані дані про виконані завдання користувача та візуалізує їх у вигляді графіків, таким чином, користувач має можливість наочно оцінити свою продуктивність.

Було реалізовано всі вимоги, що були описані у розділі 1.4. При створенні задачі, їй можна вказати назву та опис, присвоїти виконавця, тип, надати пріоритет та встановити кінцевий строк виконання. При редагуванні обраної задачі можна змінити назву, опис, пріоритет та кінцевий термін виконання. Після виконання завдання, картка перестає відображуватися на дошці, але її можна переглянути у списку виконаних задач. В даному списку виконані завдання можна переглядати, сортувати та додавати назад на дошку.

При натисканні на картку, відкривається модальне вікно, в якому користувач може переглядати список часових сесій виконання завдання, змінити статус та розпочати або припинити трекінг часу.

При додаванні користувача на дошку, можна вказати його тарифну ставку, тоді сервіс буде автоматично підраховувати заробітну плату за обраний період.

Користувач може переглянути візуалізовані дані про свою роботу за обраний проміжок часу в окремому розділі сервісу. Було реалізовано можливість переглядати такі графіки:

- гістограму, що показує кількість робочого часу в день за тиждень, місяць, рік;

- гістограму, що показує кількість виконаних завдань за тиждень, місяць, рік;
- кругову діаграму, що показує відношення кількості затраченого часу, на виконання завдань відносно їх типу;
- гістограму, що показує кількість зароблених грошей по тижням, місяцям, рокам;
- кругову діаграму, що показує відношення кількості виконаних завдань відносно їх типу;
- кругову діаграму, що показує відношення завдань що були закінчені після кінцевого строку здачі до всіх виконаних.

Також, власник дошки може переглядати візуалізовану інформацію про роботу учасників своєї дошки за обраний проміжок часу, а саме:

- гістограму, що відображає кількість робочих годин кожного працівника за спеціальністю;
- гістограму, що відображає кількість виконаних завдань одного типу працівниками однієї спеціальності;
- гістограму, що відображає відсоток завершених завдань після кінцевого строку здачі кожного працівника.

Були реалізовані вимоги інтерфейсу, що описані у підрозділі 1.4. Веб-сервіс має адаптивний user-friendly дизайн, що був протестований на реальних користувачах.

Для реалізації ієрархічної структури всередині команди було розроблено дві ролі для користувачів: «Власник» та «Користувач».

Кожен користувач має можливість переглядати яку кількість часу він витратив на виконання обраного завдання на дошці, але не може переглядати робочий час інших користувачів, окрім випадків коли його роль «Власник».

Користувачі мають можливість покидати дошку, в такому випадку право на редагування та видалення задач, що вони створили переходять

користувачу «Власник». У випадку коли дошку покидає користувач з роллю «Власник», то він має обрати кому передати свої права.

4.2. Тестування веб-сервісу

Тестування це частина розробки програмного забезпечення, що включає в себе: перевірку відповідності реалізованих функціональних можливостей до заявлених вимог, перевірку коректності роботи системи, пошук помилок та їх виправлення.

Розроблюване програмне забезпечення тестувалося протягом усього періоду розробки, тобто кожен з модулів був протестований одразу після його реалізації, що значно знизило кількість можливих помилок та ймовірність некоректної поведінки веб-сервісу.

Для тестування програмного забезпечення було обрано метод ручного тестування, його суть у тому, що тестувальник має відігравати роль кінцевого користувача щоб перевірити функції програми та її поведінку [24].

Тестування було проведено з долученням до процесу потенційних користувачів, після опрацювання їх відгуків було виділено основні зауваження до розроблюваного програмного забезпечення:

- кнопку виходу з акаунту необхідно винести зі сторінки з налаштуваннями на панель навігації;
- замінити окрему колонку з виконаними завданнями на список, що відображуватиметься за натисканням відповідної кнопки;
- додати можливість обирати картинки для аватару дошки;
- додати можливість редагувати час, що був затрачений на виконання завдання після закінчення відліку;
- додати можливість власноруч вказувати час, що був затрачений на виконання завдання;

Окремо було проведено тестування додатку на відповідність до вимог з безпеки, та виявлено, що вони реалізовані коректно: до бази даних

неможливо зробити сторонню ін’єкцію, веб-сервіс захищено від JS-ін’єкцій, програмне забезпечення має валідацію даних, введених користувачем.

Усі зібрані зауваження були опрацьовані, як наслідок, були внесені зміни у розроблюване програмне забезпечення.

Результати фінального тестування не виявили помилок у розробленому програмному забезпеченні.

4.3. Порівняння розробленого веб-сервісу з аналогами

У підрозділі 1.2 було досліджено рішення, що існують на ринку, розглянуто їх особливості, переваги та недоліки. Проведений аналіз виявив ключові функціональні можливості:

- можливість створювати задачі;
- можливість надавати задачам пріоритет та кінцевий строк виконання;
- можливість заміряти час, що витрачається на виконання завдань;

та недоліки існуючих рішень:

- складний інтерфейс;
- половина з розглянутих додатків не мала контролю за термінами виконання;
- відсутня можливість створювати командні проєкти або відсутня ієрархія всередині них;
- малий набір графіків, що візуалізують дані про виконані завдання;
- відсутній автоматичний підрахунок видатків на оплату;
- орієнтація на великі та середні ІТ-команди, що призводить до того що, програмне забезпечення має високу вартість.

Було проаналізовано різні інтерфейси для відображення завдань:

- у вигляді списку;
- у вигляді дошки;
- у вигляді календаря,

та виявлено, що найкраще сприймаються завдання, що розміщені на дошках, оскільки даний інтерфейс є найбільш наглядним, оскільки він дає змогу швидко оцінити прогрес за всіма задачами відразу.

Розроблений веб-сервіс має ключові функціональні можливості, що необхідні для програмного забезпечення для планування та трекінгу задач, широкий вибір різноманітних графіків, що відображають зібрані дані та не має недоліків, що були виявлені в ході дослідження аналогічних сервісів. Отже, розроблене програмне забезпечення може конкурувати з аналогами, що представлені на ринку.

4.4. Рекомендації щодо подальшого вдосконалення

Розроблений веб-сервіс має широкий набір інструментів, є зручним та задовольняє потреби потенційних користувачів. Оскільки область застосування розробленого програмного забезпечення має високий темп розвитку, то веб-сервіс має постійно оновлюватися щоб залишатися актуальним.

Враховуючи сучасні тенденції та відгуки цільової аудиторії було сформовано такі рекомендації до подальшого вдосконалення:

- створення окремого програмного застосунку для мобільних пристроїв з підтримкою синхронізації з веб-сервісом;
- створення системи оповіщень про нові завдання та зміни в завданнях, що виконує користувач;
- створення можливості налаштування відправлення оповіщень через месенджер Telegram;
- можливість додавати картинки у опис до картки з завданням;
- можливість додавати користувачів до дошки за посиланням;
- можливість формувати звіт про роботу команди за певний проміжок часу та зберігати його до себе на пристрій у форматі pdf-документу.

В перспективі, дане програмне забезпечення розширюється до програмного комплексу, що містить широкий функціонал та підходить не лише для трекінгу задач та заміру часу, що був витрачений на їх виконання, а і для планування цілих проєктів, що розширює область застосування даного програмного забезпечення.

4.5. Висновки до розділу

У даному розділі було проведено детальний аналіз розробленого веб-сервісу, а саме проведено опис реалізації кожної функціональної та нефункціональної вимоги, що були описані в підрозділі 1.4.

Розроблене програмне забезпечення тестувалось протягом всього періоду розробки, знайдені помилки та порушення логіки програми були виправлені. Після завершення розробки, веб-сервіс був протестований потенційними користувачами. Окремо проводилось тестування вимог до безпеки та виявлено що всі вимоги виконуються. Зауваження до реалізації функціональних можливостей та інтерфейсу, що були зібрані в ході тестування, були опрацьовані та виправлені.

Проведене порівняння розробленого програмного забезпечення з аналогами виявило що веб-сервіс має не лише ключові функціональні можливості, що необхідні для програмного забезпечення для планування та трекінгу задач, а і додатково широкий набір різноманітних графіків, що відображають зібрані дані, що призводить до того, що, розроблений веб-сервіс може конкурувати з аналогами, що представлені на ринку.

Після вивчення сучасних тенденцій, аналізу особливостей аналогічного програмного забезпечення та проведення опитування серед потенціальних користувачів, було створено перелік нововведень, що необхідні для подальшого вдосконалення та розвитку розробленого програмного застосунку.

ВИСНОВКИ

Метою даного дипломного проєкту було розроблення зручного програмного забезпечення для планування та трекінгу задач із розширеними функціональними можливостями, а саме можливістю будувати графіки, що візуально відображатимуть зібрані дані про роботу. Таке програмне забезпечення підвищить продуктивність користувачів та знизить витрати часу на аналіз виконаних задач.

Було проведено дослідження предметної галузі та детальний аналіз існуючих рішень. Враховуючи дані, зібрані під час аналізу, було створено та описано функціональні та нефункціональні вимоги, а саме вимоги до безпеки та інтерфейсу.

Для кожного з компонентів розроблюваного веб-сервісу, а саме серверної та клієнтської частин і СКБД було проаналізовано найпопулярніші засоби реалізації. Було обрано платформу Node.js для програмування серверної частини, фреймворк Vue.js для розробки клієнтської частини та MongoDB в якості СКБД.

Було створено веб-сервіс з використанням MVVM архітектури. Реалізоване програмне забезпечення має:

- сучасний та зручний графічний інтерфейс;
- таск-трекер, що представлений у вигляді дошки, на яку можна додавати картки-задачі;
- можливість надавати пріоритет, тип, опис задачам та назначати виконавця;
- можливість заміряти час, що був витрачений на виконання обраного завдання;
- ієрархічну структуру всередині проєкту;
- автоматичний підрахунок видатків на оплату;
- широкий інструментарій для візуалізації зібраних даних про роботу;
- високий рівень безпеки.

Розроблене програмне забезпечення виконане у повному обсязі та відповідає всім описаним функціональним та нефункціональним вимогам. Проведено тестування з використанням ручного способу, помилки та недоліки, що були виявлені виправлені.

Порівняння веб-сервісу з існуючими аналогами показало, що розроблене програмне забезпечення має переваги над іншими рішеннями, що робить його конкурентоспроможним на ринку.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. To Raise Productivity, Let More Employees Work from Home : [Електронний ресурс]. Режим доступу: https://hbr.org/2014/01/to-raise-productivity-let-more-employees-work-from-home
2. Harvest: Simple Online Time Tracking Software : [Електронний ресурс]. Режим доступу: https://www.getharvest.com/
3. Jira Software : [Електронний ресурс]. Режим доступу: https://www.atlassian.com/ru/software/jira
4. Timely – Fully Automatic Time Tracking : [Електронний ресурс]. Режим доступу: https://memory.ai/timely
5. Trello : [Електронний ресурс]. Режим доступу: https://trello.com/
6. C# language: [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework#c-language
7. ASP.NET Core in Action [Текст] / Andrew Loc. — Shelter Island : Manning, 2018. — 304 p.
8. About Node.js : [Електронний ресурс]. Режим доступу: https://nodejs.org/en/about/
9. Янг, Алекс. Node.js в действии [Текст] / Алекс Янг, Брэдли Мек, Майк Кантелон. — СПб. : Питер, 2015. — 448 с.
10. Express : [Електронний ресурс]. Режим доступу: https://expressjs.com/
11. About Ruby : [Електронний ресурс]. Режим доступу: https://www.ruby-lang.org/en/about/
12. Ruby on Rails: [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Ruby_on_Rails
13. Python Reference Manual : [Електронний ресурс]. Режим доступу: https://docs.python.org/2.4/ref/front.html
14. What Is React? : [Електронний ресурс]. Режим доступу: https://en.reactjs.org/tutorial/tutorial.html#what-is-react

15. Introduction to Angular concepts : [Електронний ресурс]. Режим доступу: https://angular.io/guide/architecture
16. Vue.js in Action [Текст] / E. Hanchett, B. Listwon. — Shelter Island : Manning, 2018. — 304 p.
17. DB-Engines Ranking : [Електронний ресурс]. Режим доступу: https://db-engines.com/en/ranking
18. High Performance MySQL, 3rd Edition [Текст] / Baron Schwartz, Peter Zaitsev, Vadim Tkachenko. — Sebastopol : O'Reilly Media, 2012. — 864 с.
19. Introduction to Redis : [Електронний ресурс]. Режим доступу: https://redis.io/topics/introduction
20. MongoDB: The Definitive Guide Second Edition [Текст] / Kristina Chodorow. — Sebastopol : O'Reilly Media, 2013. — 432 p.
21. Model-View-Controller в .Net : [Електронний ресурс]. Режим доступу: http://rsdn.org/article/patterns/modelviewpresenter.xml
22. Model-View-ViewModel : [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Model-View-ViewModel
23. ApexChartsJS : [Електронний ресурс]. Режим доступу: https://apexcharts.com/
24. What is Manual Testing? : [Електронний ресурс]. Режим доступу: http://www.softwaretestingclass.com/what-is-manual-testing/

ДОДАТКИ

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-СЕРВІС ДЛЯ ПЛАНУВАННЯ ТА ТРЕКІНГУ ЗАДАЧ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Уляна МАСЛЮК

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-сервіс для планування та трекінгу задач, що є веб-сайтом, серверна частина якого розроблена з використанням фреймворку Express на основі платформи Node.js, а клієнтська – з використанням фреймворку Vue.js.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) відповідність функціональних можливостей, реалізованих у програмному застосунку до заявлених;
2) коректність поведінки програмного застосунку;
3) забезпечення належного рівня безпеки даних;
4) зручність роботи з веб-сервісом.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування проводитиметься на рівні «системного тестування», тобто буде перевірено кожен модуль розробленого програмного продукту та зв’язки між ними. Для проведення тестування було обрано метод White Box Testing. При такому методі перевіряється як поведінка програмного продукту, так і код.

Використовуються наступні методи:

1) Для проведення функціонального тестування Critical path test (тестування критичного шляху);
2) Для тестування продуктивності програмного забезпечення load testing (навантажувальне тестування) та stress testing (стресс-тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Коректність роботи веб-сервісу тестується з використанням таких методів та утиліт:

1) введенням недопустимих значень в поля, що можна редагувати;
2) ручного тестування відповідності реалізованих функціональних вимог до заявлених;
3) тестування при максимальному навантаженні за допомогою утиліти Apache JMeter;
4) тестування стабільності роботи при різних умовах за допомогою утиліти Apache JMeter;
5) тестування веб-сервісу в різних браузерах;
6) тестування зручності користувацького інтерфейсу опитуванням потенційних користувачів.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-СЕРВІС ДЛЯ ПЛАНУВАННЯ ТА ТРЕКІНГУ ЗАДАЧ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Уляна МАСЛЮК

2020

ЗМІСТ

1. Опис структури веб-сервісу

Веб-сервіс для аналізу та трекінгу задач складається з наступних сторінок:

- «Login»;
- «Registration»;
- «My account»;
- «My boards»;
- «Analytics».

Перехід між сторінками відбувається за натисканням відповідних кнопок на боковій навігаційній панелі.

2. Процедура авторизації користувача

Сторінка «Login» відкривається за змовчуванням для неавторизованих користувачів (рис. 1). Сторінка містить форму для вводу необхідних для авторизації даних: юзернейма та пароля. Після введення коректних даних та натиснення кнопки «LOGIN», користувач увійде в систему. Також, на сторінці міститься посилання на сторінку «Register».

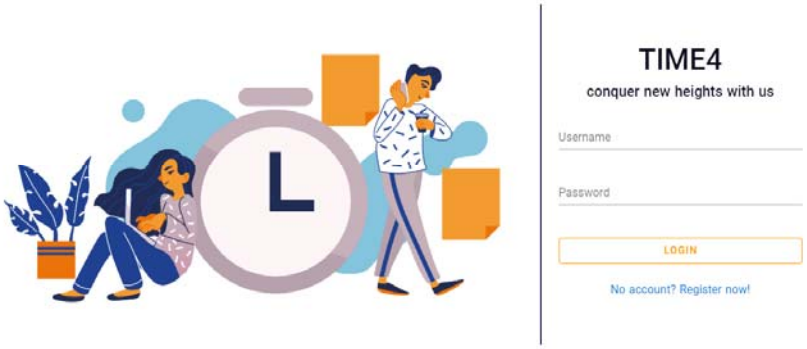


Рис. 1. Сторінка «Login»

3. Процедура реєстрації користувача

Сторінка «Register» (рис. 2) містить форму для вводу необхідних для реєстрації даних: адресу поштової скриньки, юзернейм, пароль, ім’я, прізвище, дату народження та місто проживання. Після введення коректних даних та натиснення кнопки «REGISTER», користувач увійде в систему. Також, на сторінці міститься посилання на сторінку «Login».

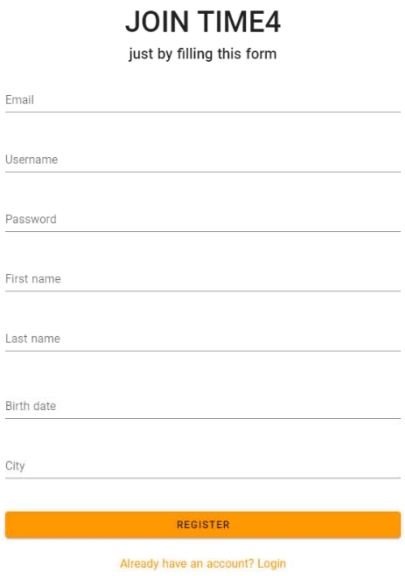


Рис. 2. Сторінка «Register»

4. Сторінка «User» та редагування даних про акаунт

Після успішної авторизації, користувач потрапляє на сторінку «User» (рис. 3). На цій сторінці містяться аватар користувача та інформація про нього. При натисканні на значок із шестернею, відкривається модальне вікно (рис. 4) в якому користувач може редагувати дані про себе.

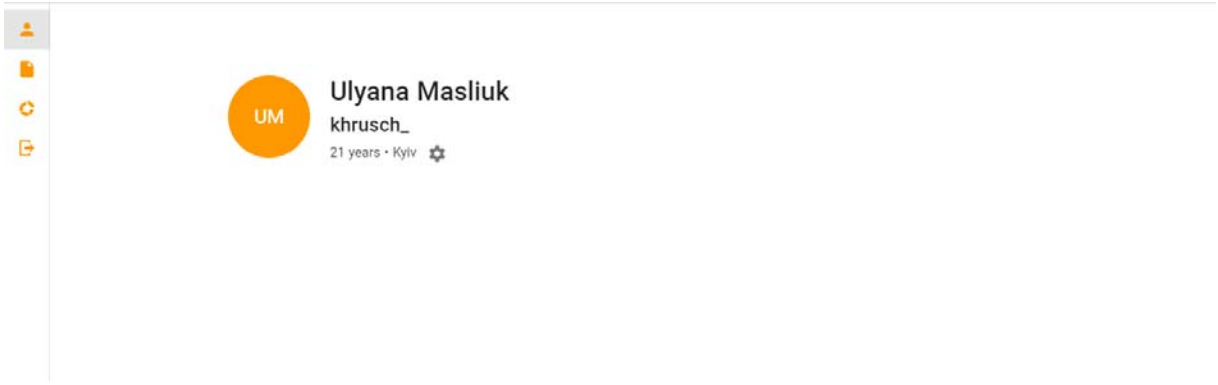


Рис. 3. Сторінка «User»

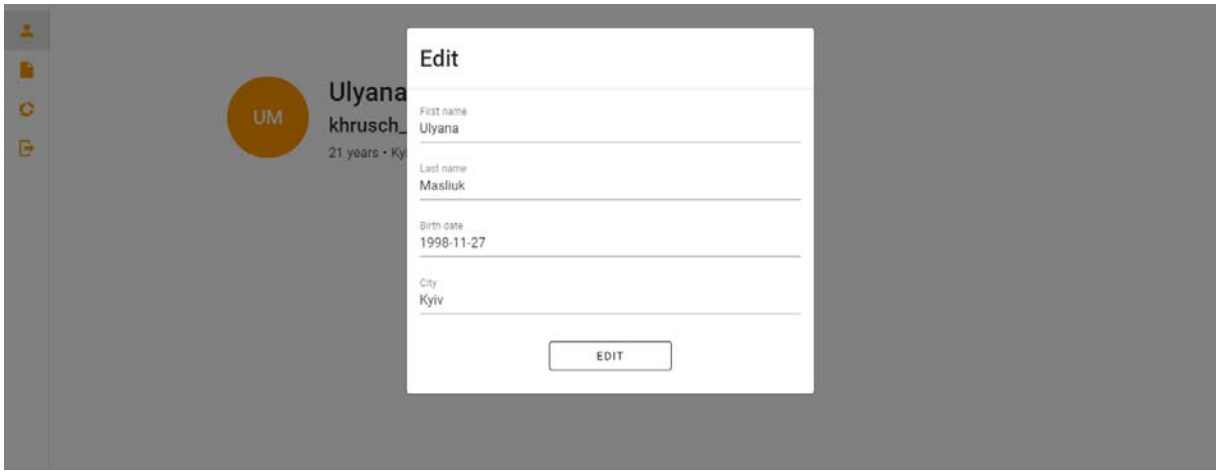


Рис. 4. Модальне вікно для зміни інформації про акаунт

5. Створення нової дошки

Авторизований користувач може переглядати свої дошки на сторінці «Boards» (рис. 5). На цій сторінці містяться дошки, до яких корситувач має доступ та дві кнопки: «CREATE NEW BOARD» та «JOIN BOARD». Після натискання на першу кнопку, відкриється модульне вікно (рис. 6), куди користувач може внести назву нової дошки.

Рис. 5. Сторінка «Boards»

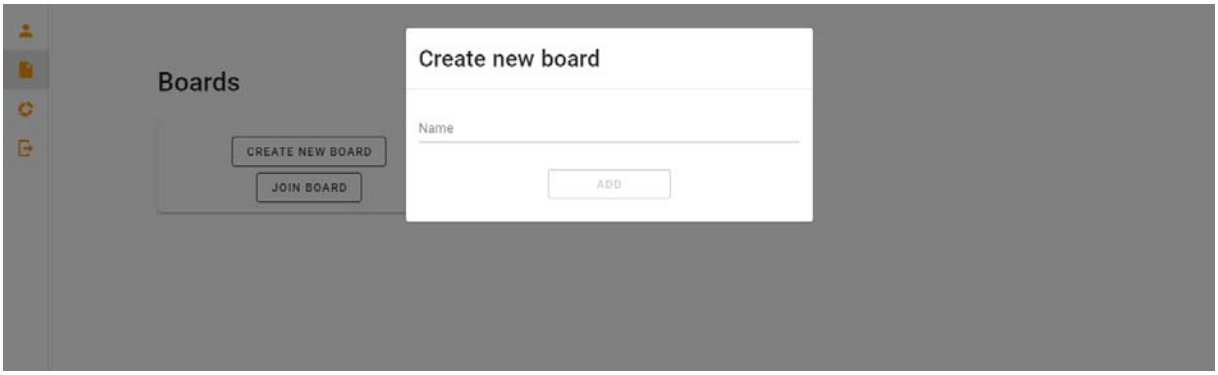


Рис. 6. Модальне вікно для створення нової дошки

6. Приєднання до дошки за логіном та паролем

Авторизований користувач може переглядати свої дошки на сторінці «Boards» (рис. 5). На цій сторінці містяться дошки, до яких корситувач має доступ та дві кнопки: «CREATE NEW BOARD» та «JOIN BOARD». Після натискання на другу кнопку, відкриється модульне вікно (рис. 7), куди користувач може ввести ідентифікатор та пароль від вже створеної дошки, таким чином, приєднавшись до неї.

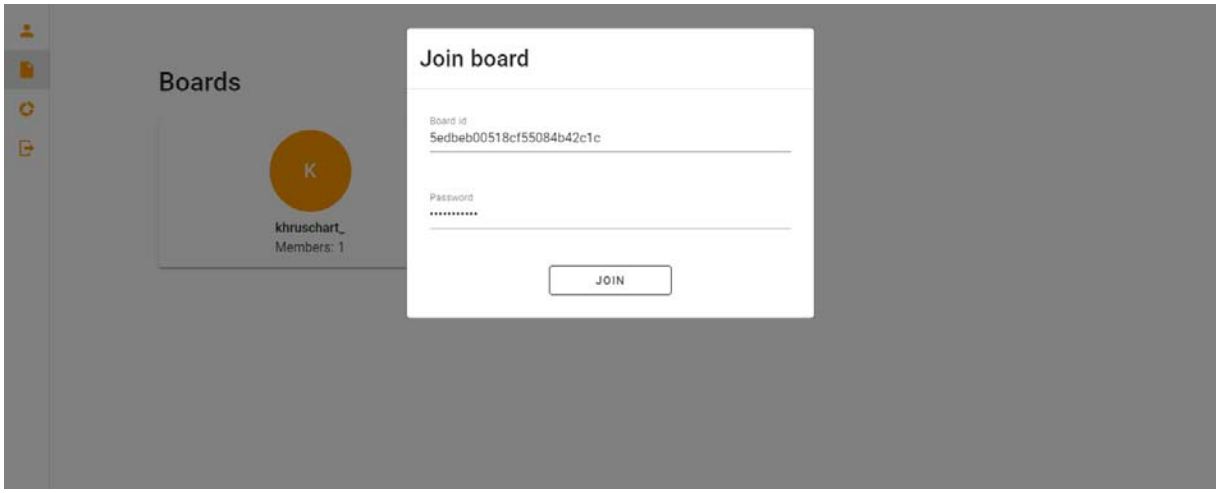


Рис. 7. Модальне вікно для приєднання до існуючої дошки

7. Створення, редагування та видалення колонок

Після вибору дошки на сторінці «Boards», користувач потрапляє на її сторінку (рис. 8), на якій знаходяться колонки з задачами-картками. При натисканні на кнопку «ADD COLUMN», відкривається модальне вікно (рис. 9), куди можна ввести назву нової колонки. Після натискання на кнопку «ADD» нова колонка буде створена.

Для редагування колонки необхідно натиснути на кнопку з трьома вертикальними крапками,тоді відкриється модальне вікно (рис. 10) в якому можна задати нові дані: назву та позицію у списку, або видалити колонку. При видаленні колонки, всі завдання, що містяться в ній будуть також видалені, про що користувач отримає відповідне попередження (рис. 11).

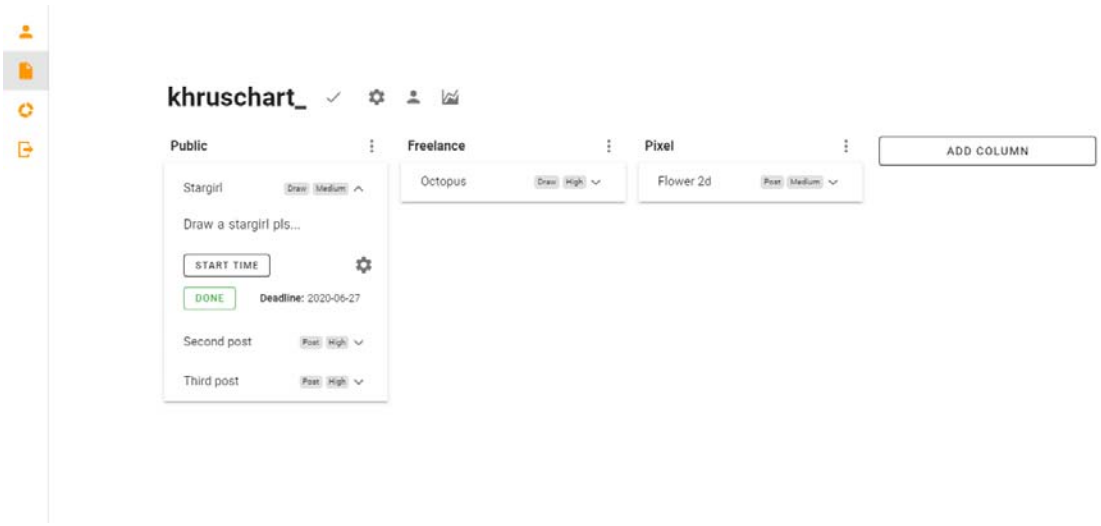


Рис. 8. Сторінка дошки «khruschart_»

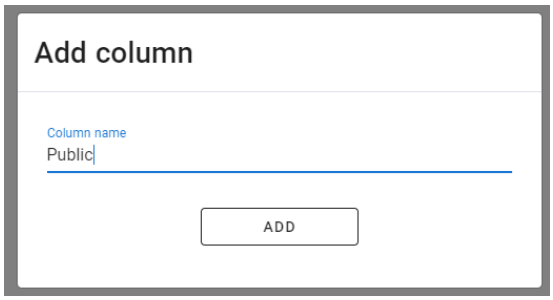


Рис. 9. Модальне вікно для створення нової колонки

Рис. 10. Модальне вікно для редагування колонки

Рис. 11. Попередження перед видаленням колонки

8. Створення, редагування та видалення карток-задач

На сторінці «Boards» знаходяться колонки з задачами-картками. При натисканні на круглу кнопку з плюсиком, відкривається модальне вікно (рис. 12), куди можна ввести назву, описання, тип, пріоритет, кінцевий строк виконання та якій колонці буде належати нова задача. Після натискання на кнопку «ADD» задача буде створена.

Створені задачі можна редагувати. Для цього необхідно натиснути на кнопку з шестернею на розгорнутому завданні. Відкриється модальне вікно (рис. 13), в якому користувач може редагувати дані про завдання. При натисканні на кнопку «DELETE» завдання буде видалено.

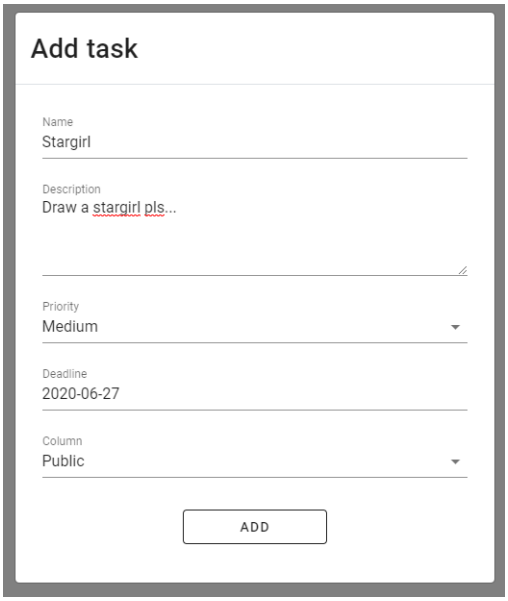


Рис. 12. Модальне вікно для створення нової задачі

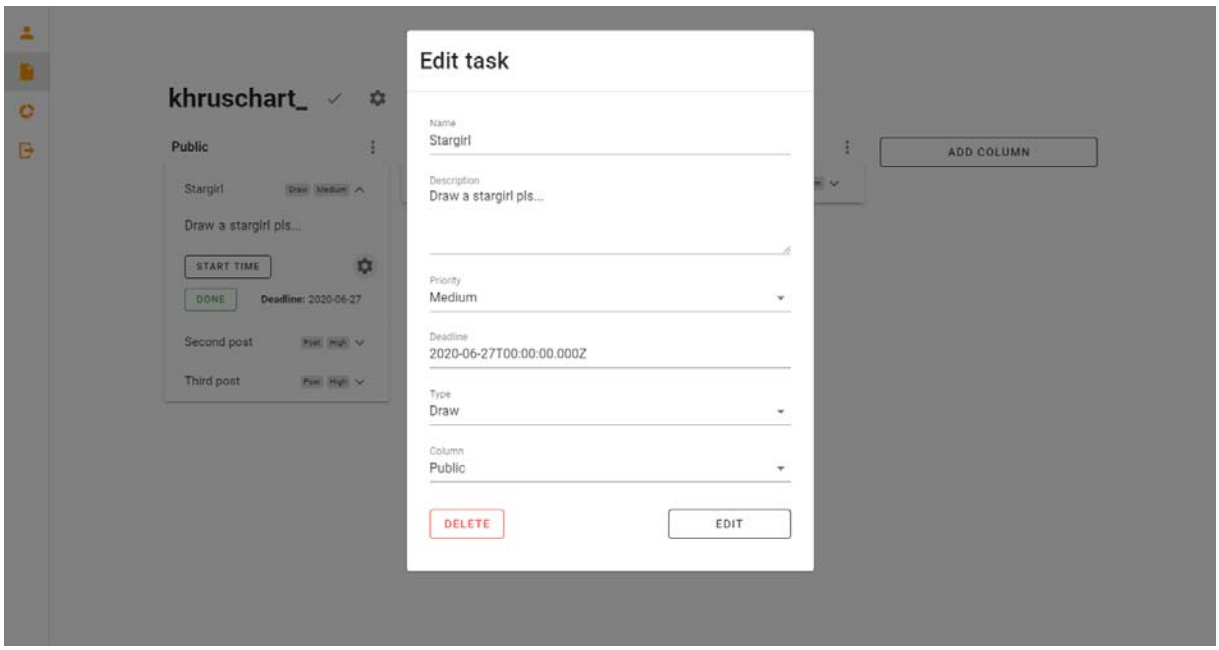


Рис. 13. Модальне вікно для редагування задачі

9. Заміряння часу, витраченого на задачу, список «Done»

На сторінці «Boards» знаходяться колонки з задачами-картками. При розгортанні картки у списку, відображається інформація про цю задачу: назва, опис, тип, пріоритет, кількість витраченого часу та кнопки «START TIME» та «DONE» (рис. 14). При натисканні на першу кнопку, починається відлік часу та відображується кнопка «END», натиснувши котру відлік часу зупиниться (рис. 15).

При натисканні на кнопку «DONE», задача переміщається до списку «Done». Для того, щоб переглянути список «Done» треба натиснути на кнопку з галочкою біля назви дошки. Після натискання кнопки, відобразиться список виконаних завдань (рис. 16).

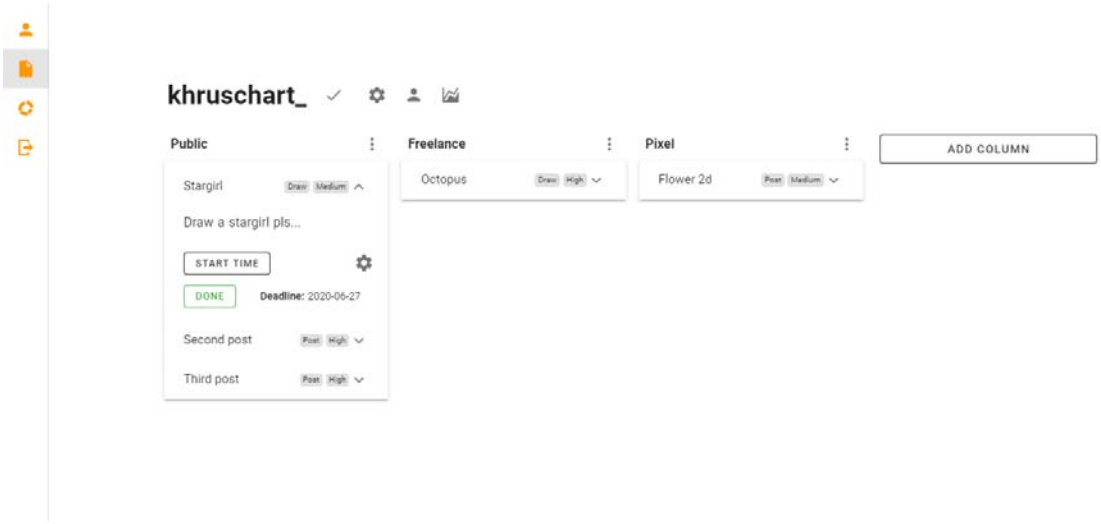


Рис. 14. Розгорнута картка з завданням «Stargirl»

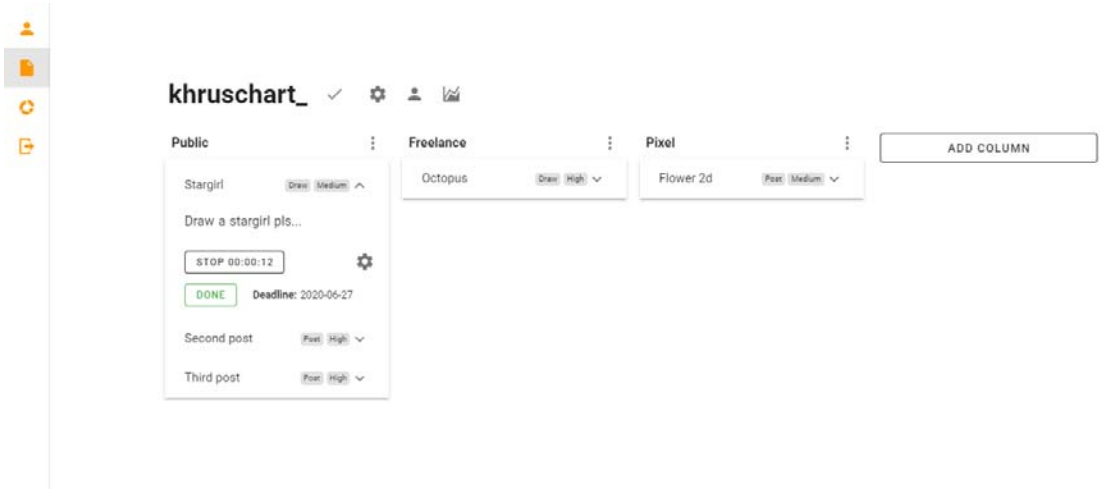


Рис. 15. Запущено відлік часу на завданні «Stargirl»

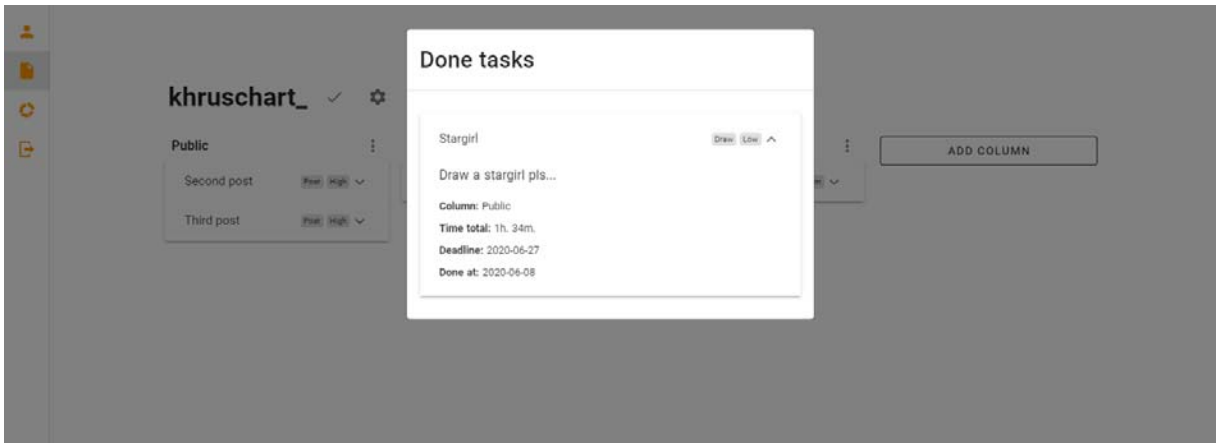


Рис. 16. Модальне вікно зі списком виконаних задач

10. Панель адміністрування дошки

Якщо роль користувача «Власник», то на сторінці «Boards» відображуються кнопки з шестернею, графіком та силуетом. При натисканні на кнопку з шестернею, користувачеві відображується модальне вікно, на якому користувач має змогу змінити назву та типи задач на дошці (рис. 17). При натисканні на кнопку з графіком, користувач має можливість переглянути візуалізовані дані про роботу інших членів дошки (рис. 18). При натисканні на кнопку з силуетом, користувачеві відображається список членів команди, а також board id та password, що необхідні для приєднання до дошки (рис. 19).

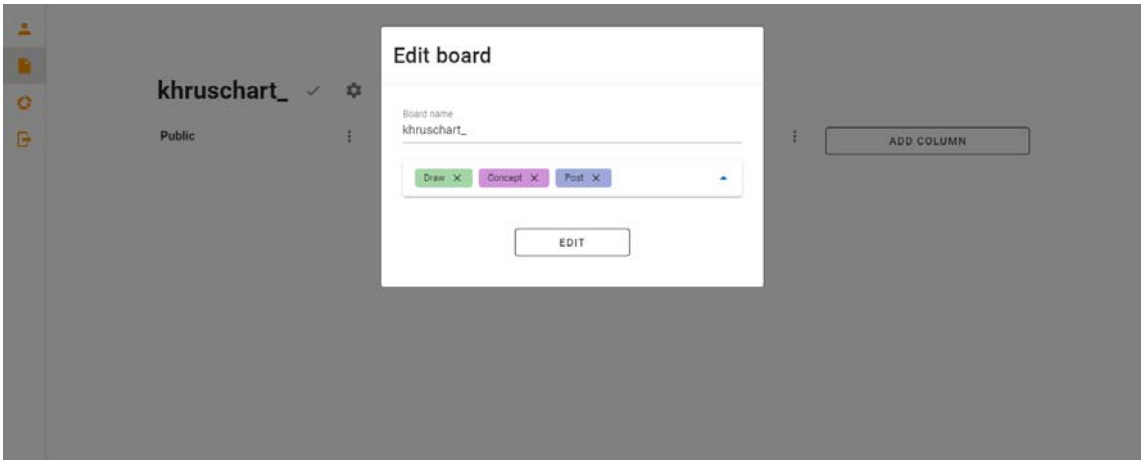


Рис. 17. Модальне вікно для редагування даних про дошку

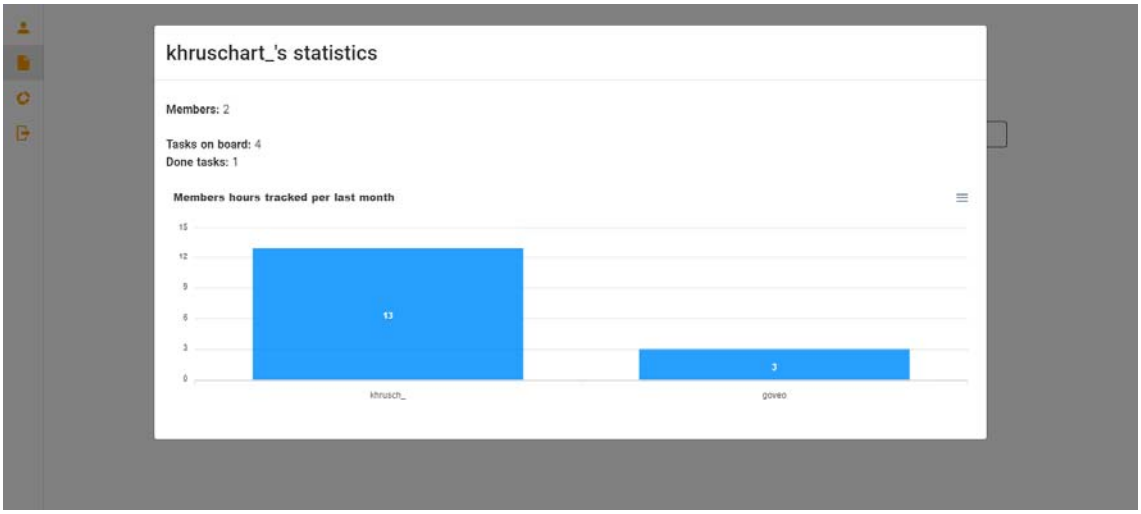


Рис. 18. Модальне вікно з графіками про роботу членів команди

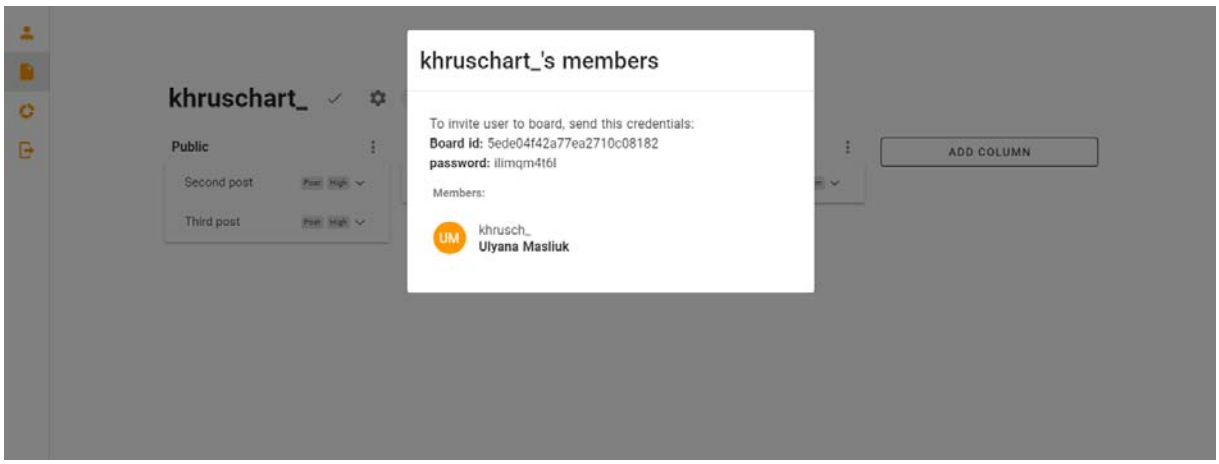


Рис. 19. Модальне вікно зі списком членів дошки, ідентифікатором та паролем

11. Сторінка з візуалізованими даними про задачу

На сторінці «Analytics» (рис. 20) відображуються візуалізовані дані про роботу користувача, наприклад кількість часу витрачена на різних дошках та кругова діаграма, що відображає відношення часу витраченого на виконання задач відносно типу.

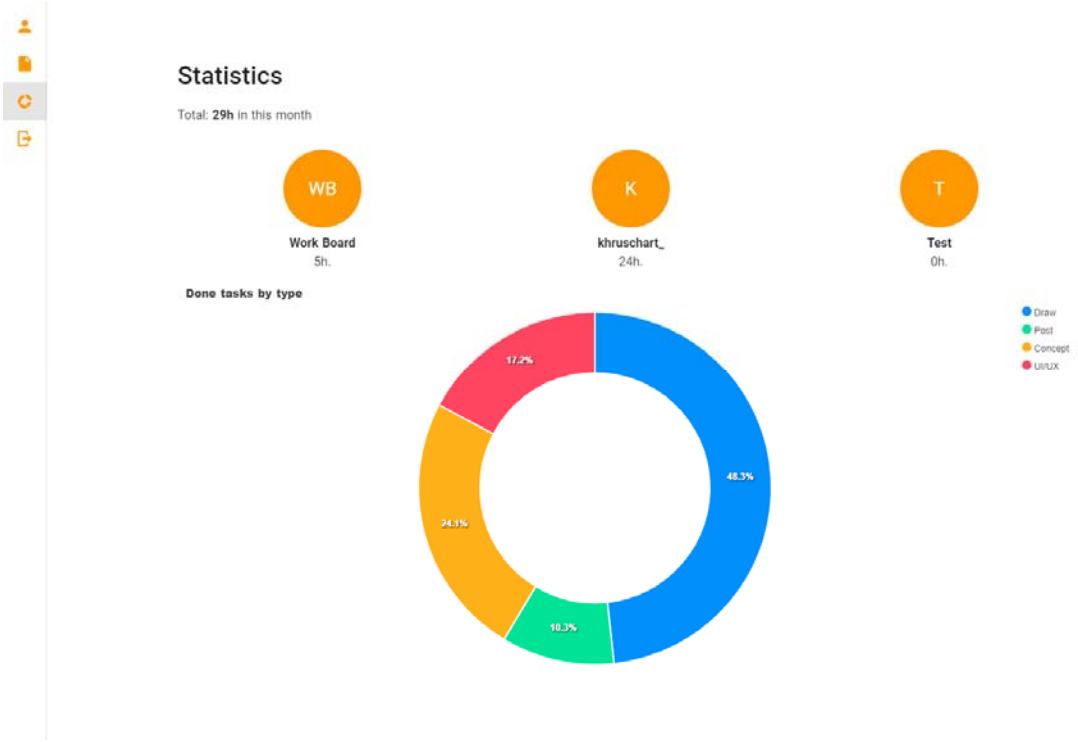


Рис. 20. Сторінка «Analytics»

12. Вихід з акаунту

Після натиснення на нижню кнопку вертикальної навігаційної панелі відбувається вихід з акаунту (рис. 21).

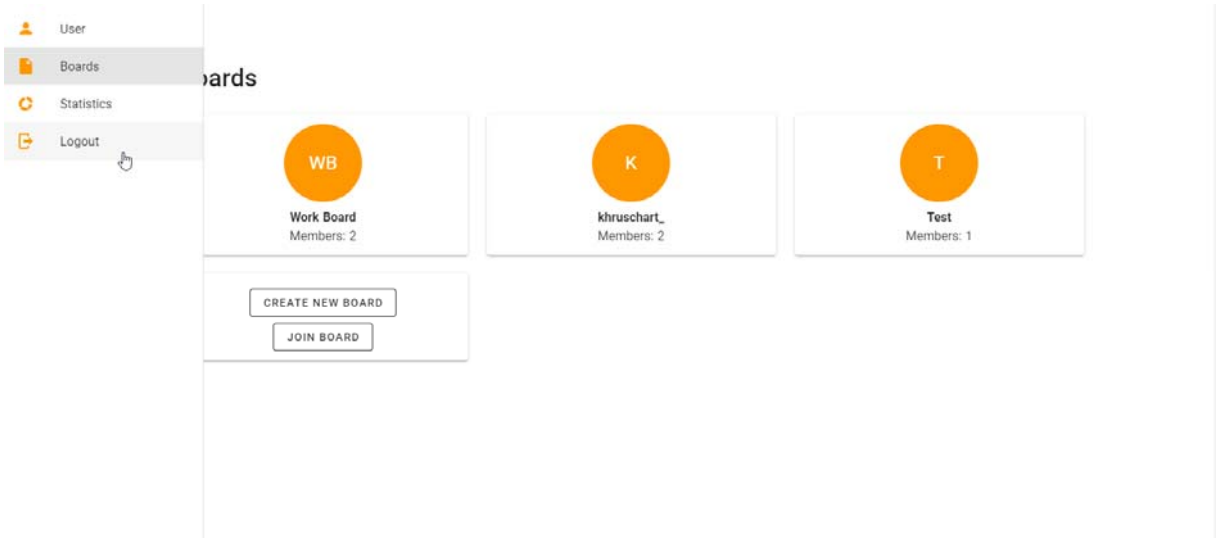
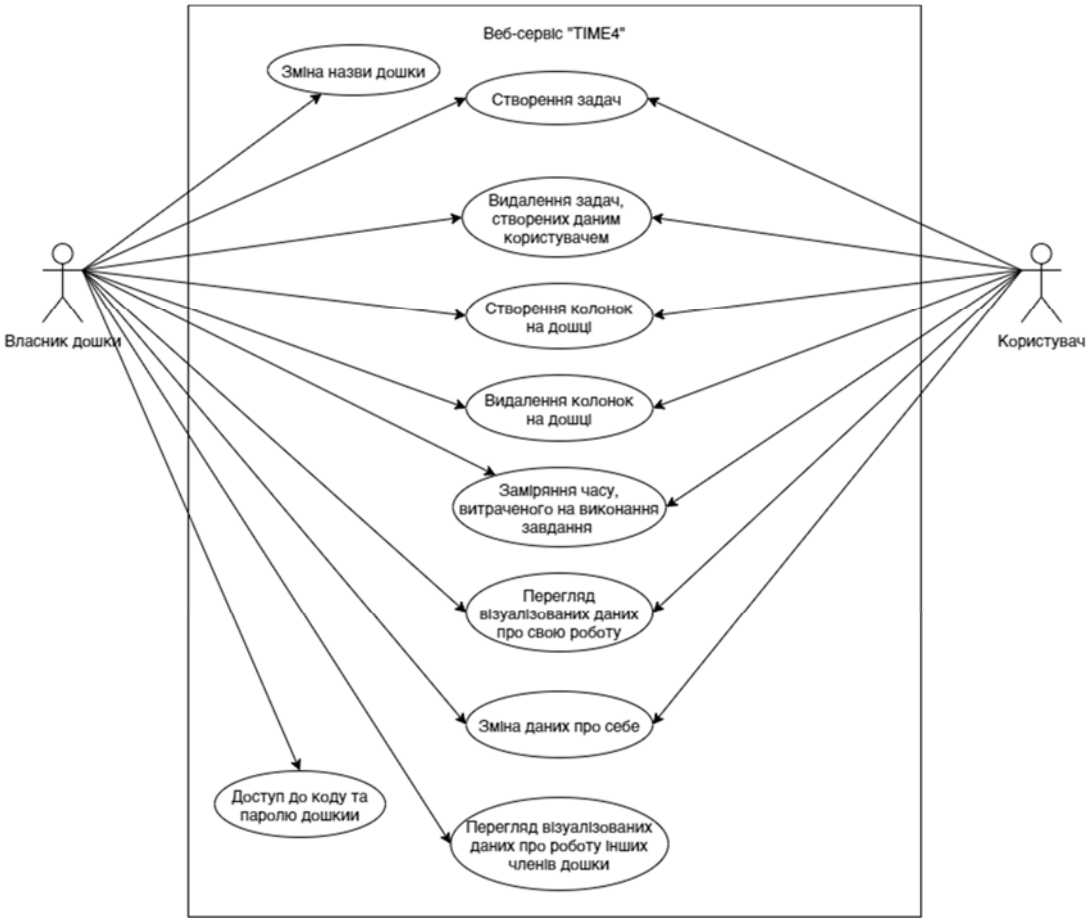


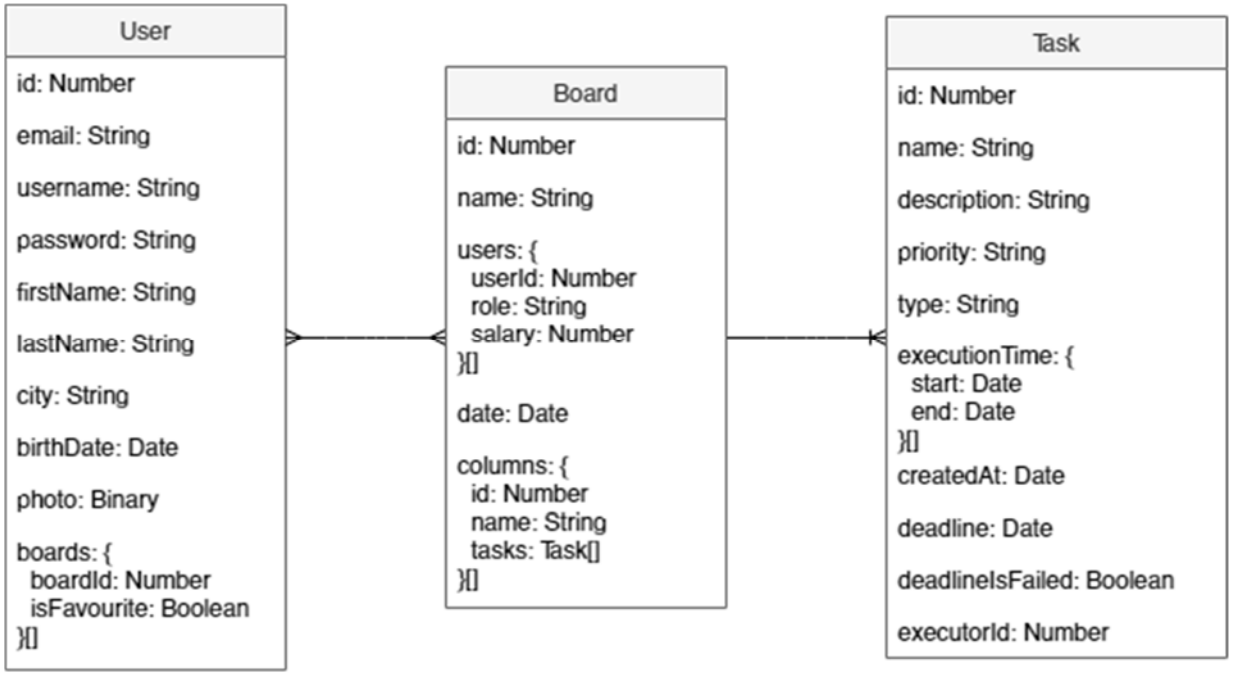
Рис. 21. Вихід з акаунту

Додаток 1

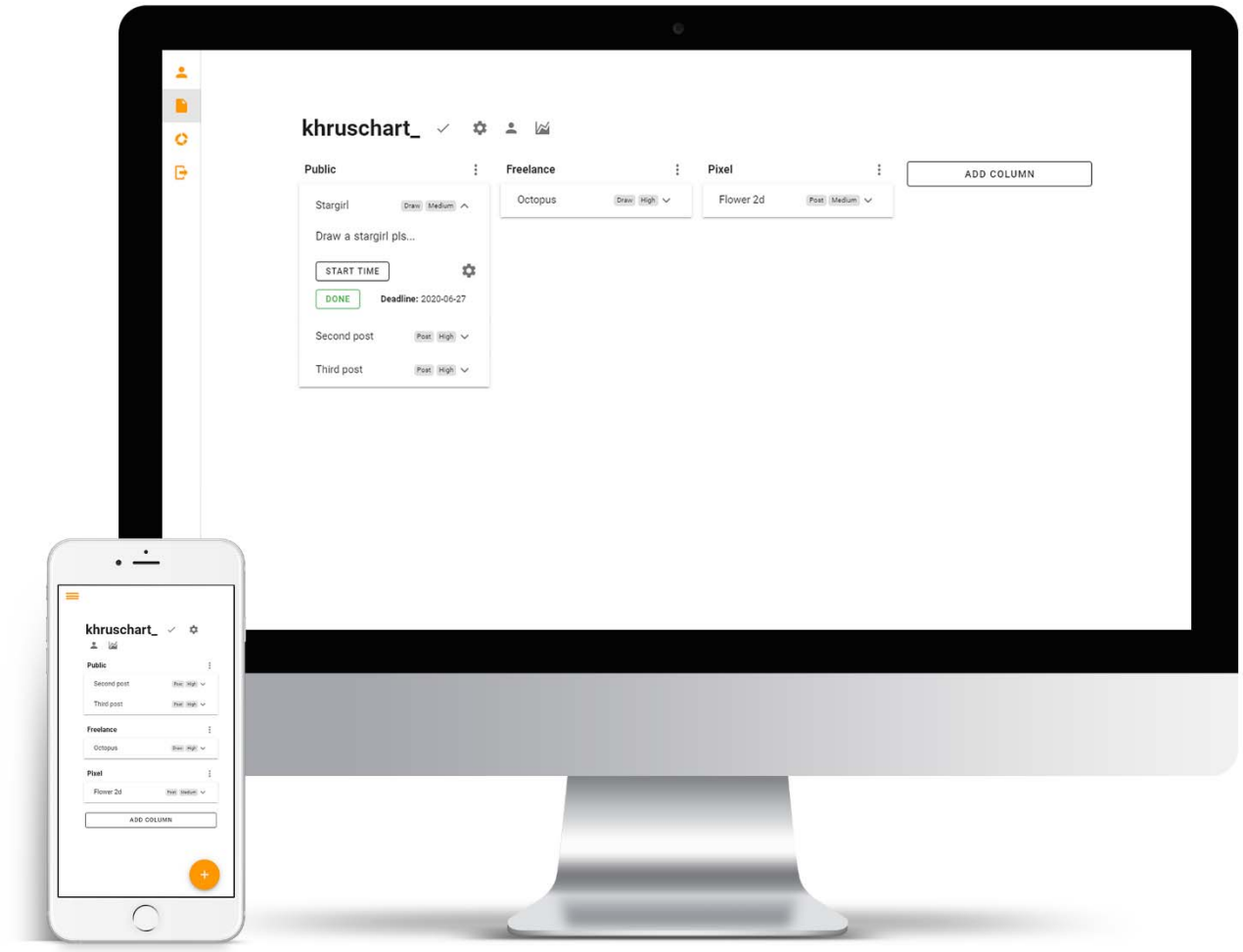
Копії графічних матеріалів



ДП.045440-06-99.
Веб-сервіс для планування та трекінгу задач.
Діяльність користувача у системі.
Діаграма використання



ДП.045440-07-99.
Веб-сервіс для планування та трекінгу задач.
Архітектура системи. Схема бази даних веб-сервісу. ER-діаграма



Маслюк Уляна Олександрівна
гр. КП-61

Маслюк Уляна Олександрівна
гр. КП-61

Додаток 2

Текст програми

2.1. Модуль для авторизації

```
const crypto = require('crypto');
const User = require('../database/models/user');
const config = require('../../app.config');
const passportJWT = require('passport-jwt');
const passportLocalStrategy = require('passport-local').Strategy;
const passportJWTStrategy = passportJWT.Strategy;
const ExtractJWT = passportJWT.ExtractJwt;
const serverSalt = config.SALT;

function sha512(password) {
  const hash = crypto.createHmac('sha512', serverSalt);
  hash.update(password);
  const value = hash.digest('hex');
  return {
    passwordHash: value,
  };
}

function serializeUser(user, done) {
  done(null, user._id);
}

async function deserializeUser(id, done) {
  try {
    const user = await User.getById(id);
    done(null, user);
  }
  catch (err) {
    done(err, null);
  }
}

const LocalStrategy = new passportLocalStrategy(async (username,
password, done) => {
  try {
    const hashedPass = sha512(password, serverSalt).passwordHash;
    const user = await User.getByCredentials(username, hashedPass);
    if (!user) done(null, false, { message: 'Incorrect username or
password' });
    else done(null, user, { message: 'Logged In Successfully' });
  }
  catch (err) {
    done(err, null, { message: 'Some database error' });
  }
});

const JWTStrategy = new passportJWTStrategy({
  jwtFromRequest: ExtractJWT.fromAuthHeaderWithScheme('jwt'),
  secretOrKey: config.JWT_SECRET,
}, async (jwtPayload, done) => {
```

```
  try {
    const user = await User.getById(jwtPayload.id);
    if (user) {
      done(null, user, { message: 'Logged in successfully' });
    }
    else {
      done(null, false, { message: 'Incorrect username or password'
});
    }
  }
  catch (err) {
    done(err);
  }
});

function verifyToken(req, res, next) {
  const bearerHeader = req.headers['authorization'];
  if (bearerHeader !== undefined) {
    const bearer = bearerHeader.split(' ');
    const bearerToken = bearer[1];
    req.token = bearerToken;
    next();
  }
  else {
    res.status(401).json({ error: 'Unauthorized' });
  }
}

module.exports = {
  sha512,
  verifyToken,
  serializeUser,
  deserializeUser,
  LocalStrategy,
  JWTStrategy,
};
```

2.2. Схема сутності «Board»

```
const mongoose = require('mongoose');

const BoardSchema = mongoose.Schema({
  name: String,
  password: {
    type: String,
    default: Math.random().toString(36).slice(2),
  },
  ownerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  users: [{
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    role: String,
```

```
    salary: Number,
  }],
  taskTypes: [String],
  columns: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Column' }],
}, { timestamps: true });


const BoardModel = mongoose.model('Board', BoardSchema);

class Board {
  constructor({
    ownerId, name, users=[{
      user: {
        _id: ownerId,
        role: null,
        salary: null,
      },
    }],
    taskTypes=[],
    columns=[],
  }) {
    this.ownerId = ownerId;
    this.name = name;
    this.users = users;
    this.taskTypes = taskTypes;
    this.columns = columns;
  }

  static Model = BoardModel;

  static getById(id) {
    return BoardModel.findById(id)
      .populate({
        path: 'columns',
        options: { sort: { 'index': 1 } },
        populate: {
          path: 'tasks',
          options: { sort: { 'index': 1 } },
        },
      })
      .populate({
        path: 'users.user',
      });
  }

  static getByOwnerId(ownerId) {
    return BoardModel.find({ ownerId });
  }

  static getByUserId(userId) {
```

```
    return BoardModel.find({ users: { $elemMatch: { user: userId } }});
  }

  static getByIdAndUserId(id, userId) {
    return BoardModel.findOne({ _id: id, users: { $elemMatch: { user: userId } }});
  }

  static getByIdAndPassword(id, password) {
    return BoardModel.findOne({
      _id: id,
      password,
    });
  }

  static insert(board) {
    return new BoardModel(board).save();
  }

  static update(id, board) {
    return BoardModel.updateOne({ _id: id }, board);
  }

  static delete(id) {
    return BoardModel.deleteOne({ _id: id });
  }

  static getColumnsCount(boardId) {
    return BoardModel.aggregate([
      {
        $match: { _id: mongoose.Types.ObjectId(boardId) },
      },
      {
        $project: {
          columns: {
            $size: '$columns',
          },
        },
      },
    ]);
  }

  static addColumn(boardId, columnId) {
    return BoardModel.updateOne({ _id: boardId },
      {
        $addToSet: {
          columns: columnId,
        },
      },
    );
```

```
  }

  static removeColumn(boardId, columnId) {
    return BoardModel.updateOne({ _id: boardId },
      {
        $pull: {
          columns: columnId,
        },
      },
    );
  }

  static addUser(boardId, userId) {
    return BoardModel.updateOne({ _id: boardId },
      {
        $addToSet: {
          users: {
            user: userId,
            salary: null,
            role: null,
          },
        },
      },
    );
  }

  static addTaskType(boardId, type) {
    return BoardModel.updateOne({ _id: boardId },
      {
        $addToSet: {
          taskTypes: type,
        },
      },
    );
  }
}


module.exports = Board;
```

2.3. Схема сутності «Column»

```
const mongoose = require('mongoose');


const ColumnSchema = mongoose.Schema({
  boardId: { type: mongoose.Schema.Types.ObjectId, ref: 'Board' },
  name: String,
  index: Number,
  tasks: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Task' }],
```

```
}, { timestamps: true });

const ColumnModel = mongoose.model('Column', ColumnSchema);

class Column {
  constructor({
    boardId, name, index, tasks=[],
  }) {
    this.boardId = boardId;
    this.name = name;
    this.index = index;
    this.executorId = tasks;
  }

  static Model = ColumnModel;

  static getById(id) {
    return ColumnModel.findById(id).populate({
      path: 'tasks',
    });
  }

  static getByBoardId(boardId) {
    return ColumnModel.find({ boardId });
  }

  static insert(column) {
    return new ColumnModel(column).save();
  }

  static update(id, column) {
    return ColumnModel.updateOne({ _id: id }, column);
  }
```

```
static async updateIndex(boardId, columnId, index) {
  const column = await ColumnModel.findById(columnId);
  const currentIndex = column.index;
  column.index = index;
  if (currentIndex > index) {
    await ColumnModel.updateMany(
      {
        boardId: mongoose.Types.ObjectId(boardId),
        index: {
          $gte: index,
          $lt: currentIndex,
        },
      },
      {
        $inc: { index: 1 },
      },
    );
  }
  else {
    await ColumnModel.updateMany(
      {
        boardId: mongoose.Types.ObjectId(boardId),
        index: {
          $gt: currentIndex,
          $lte: index,
        },
      },
      {
        $inc: { index: -1 },
      },
    );
  }
  return ColumnModel.updateOne({ _id: columnId }, column);
}
```

```
static addTask(columnId, taskId) {
  return ColumnModel.updateOne({ _id: columnId },
    {
      $push: {
        tasks: taskId,
      },
    },
  );
}

static deleteTask(columnId, taskId) {
  return ColumnModel.updateOne({ _id: columnId },
    {
      $pull: {
        tasks: taskId,
      },
    },
  );
}

static getTasksCount(columnId) {
  return ColumnModel.aggregate([
    {
      $match: { _id: mongoose.Types.ObjectId(columnId) },
    },
    {
      $project: {
        tasks: {
          $size: '$tasks',
        },
      },
    },
  ]);
```

```
  }

  static async delete(boardId, columnId) {
    const column = await ColumnModel.findById(columnId);
    const currentIndex = column.index;

    await ColumnModel.updateMany(
      {
        boardId: mongoose.Types.ObjectId(boardId),
        index: {
          $gt: currentIndex,
        },
      },
      {
        $inc: { index: -1 },
      },
    );

    return ColumnModel.deleteOne({ _id: columnId });
  }
}

module.exports = Column;
```

2.4. Схема сутності «Task»

```
const mongoose = require('mongoose');

const TaskSchema = mongoose.Schema({
  boardId: { type: mongoose.Schema.Types.ObjectId, ref: 'Board' },
  columnId: { type: mongoose.Schema.Types.ObjectId, ref: 'Column' },
  index: Number,
  name: String,
  description: String,
  priority: {
```

```
    type: String,
    default: 'Medium',
  },
  executionTime: [{
    start: Date,
    end: Date,
  }],
  deadline: Date,
  type: String,
  executorId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
}, { timestamps: true });

const TaskModel = mongoose.model('Task', TaskSchema);

class Task {
  constructor({
    boardId, columnId, index, name, description=null, priority=1,
executionTime=[], deadline=null, type=null, executorId=null,
  }) {
    this.boardId = boardId;
    this.columnId = columnId;
    this.index = index;
    this.name = name;
    this.description = description;
    this.priority = priority;
    this.executionTime = executionTime;
    this.deadline = deadline;
    this.type = type;
    this.executorId = executorId;
  }

  static Model = TaskModel;

  static getById(id) {
```

```
  return TaskModel.findById(id);
}

static getByExecutorId(executorId) {
  return TaskModel.find({ executorId });
}

static getByColumnId(columnId) {
  return TaskModel.find({ columnId });
}

static insert(task) {
  return new TaskModel(task).save();
}

static update(id, task) {
  return TaskModel.updateOne({ _id: id }, task);
}

static async updateIndex(columnId, taskId, index) {
  const task = await TaskModel.findById(taskId);
  const currentIndex = task.index;
  task.index = index;
  if (currentIndex > index) {
    await TaskModel.updateMany(
      {
        columnId: mongoose.Types.ObjectId(columnId),
        index: {
          $gte: index,
          $lt: currentIndex,
        },
      },
      {
        $inc: { index: 1 },
```

```
      },
    );
  }
  else if (currentIndex < index) {
    await TaskModel.updateMany(
      {
        columnId: mongoose.Types.ObjectId(columnId),
        index: {
          $gt: currentIndex,
          $lte: index,
        },
      },
      {
        $inc: { index: -1 },
      },
    );
  }
  return TaskModel.updateOne({ _id: taskId }, task);
}

static async delete(columnId, taskId) {
  const column = await TaskModel.findById(taskId);
  const currentIndex = column.index;

  await TaskModel.updateMany(
    {
      columnId: mongoose.Types.ObjectId(columnId),
      index: {
        $gt: currentIndex,
      },
    },
    {
      $inc: { index: -1 },
    },
```

```
    );

    return TaskModel.deleteOne({ _id: taskId });
  }
}

module.exports = Task;
```

2.5. Схема сутності «User»

```
const mongoose = require('mongoose');
const autoIncrement = require('mongoose-auto-increment');
autoIncrement.initialize(mongoose);

const UserSchema = mongoose.Schema({
  email: {
    type: String,
    unique: true,
  },
  username: {
    type: String,
    unique: true,
  },
  password: String,
  firstName: String,
  lastName: String,
  birthDate: Date,
  city: String,
  boardsIds: {
    boardsId: Number,
    isFavorite: Boolean,
  },
}, { timestamps: true });

UserSchema.plugin(autoIncrement.plugin, { model: 'User', field: 'id'
});

const UserModel = mongoose.model('User', UserSchema);

class User {
  constructor({
    email, username, password, firstName, lastName, birthDate=null,
city='', boardsIds=[],
  }) {
    this.email = email;
    this.username = username;
    this.password = password;
```

```
    this.firstName = firstName;
    this.lastName = lastName;
    this.birthDate = birthDate;
    this.city = city;
    this.boardsIds = boardsIds;
  }

  static validUsername(str) {
    return typeof str === 'string' && /^[a-zA-Z0-9_]+$/.test(str)
      && str.trim().length >= 5 && str.trim().length <= 16;
  }

  static validEmail(str) {
    // eslint-disable-next-line no-control-regex
    return typeof str === 'string' && /(?:[a-z0-9!#$%&'*+/=?^_`{|}~-
]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-
\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-
\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-
]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-
9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])|[a-z0-9-
]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-
\x09\x0b\x0c\x0e-\x7f])+)\])/.test(str);
  }

  static validName(str) {
    return typeof str === 'string' && /^[a-яА-ЯёЁa-zA-
Z\s]+$/.test(str)
      && str.trim().length >= 1 && str.trim().length <= 32;
  }

  static validCity(str) {
    return typeof str === 'string' && /^[a-яА-ЯёЁa-zA-
Z\s]+$/.test(str)
      && str.trim().length >= 1 && str.trim().length <= 32;
  }

  static getAll() {
    return UserModel.find({});
  }

  static getById(id) {
    return UserModel.findById(id);
  }

  static getByUsername(username) {
    return UserModel.findOne({ username });
  }

  static getByCredentials(username, password) {
    return UserModel.findOne({ username, password });
  }
```

```
  static insertUser(user) {
    if (!this.validateUser(user)) {
      return Promise.reject(new Error('User object failed
validation.'));
    }
    else {
      return new UserModel(user).save();
    }
  }

  static update(id, user) {
    return UserModel.updateOne({ _id: id }, user);
  }

  static delete(id) {
    return UserModel.deleteOne({ _id: id });
  }

  static validateUser(user) {
    return this.validUsername(user.username);
  }

}

module.exports = User;
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-СЕРВІС ДЛЯ ПЛАНУВАННЯ ТА ТРЕКІНГУ ЗАДАЧ

Виконала: Маслюк Уляна Олександрівна

Керівник: ст. викладач, к.т.н., Люшенко Леся Анатоліївна

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: створення програмного забезпечення, що надає функціональні можливості для планування та трекінгу задач та часу, витраченого на їх виконання та подальшого аналізу даних про виконані задачі у вигляді графіків, побудованих за різними критеріями.

Завдання:

1. Проаналізувати предметну область, дослідити існуючі рішення на ринку та визначити вимоги до розроблення програмного додатку.
2. Розробити програмне забезпечення відповідно до зібраних вимог.
3. Протестувати розроблене програмне забезпечення.

АКТУАЛЬНІСТЬ

ЦІЛЬОВА АУДИТОРІЯ

Фрилансери

Невеликі IT-команди

Люди, що займаються тайм-менеджментом

ІСНУЮЧІ РІШЕННЯ

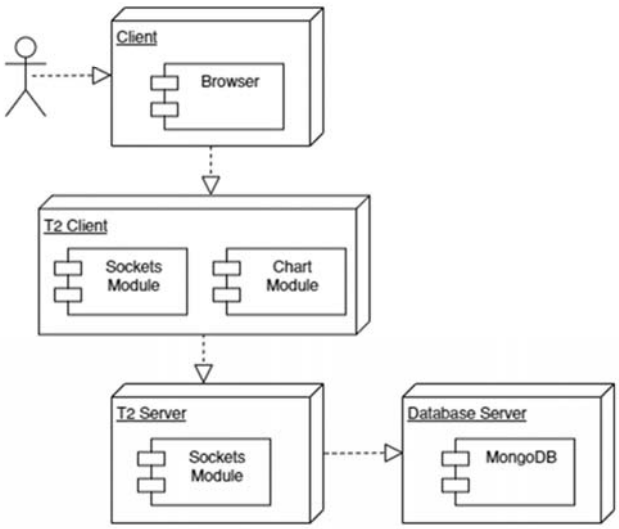
Trello

HARVEST

Jira

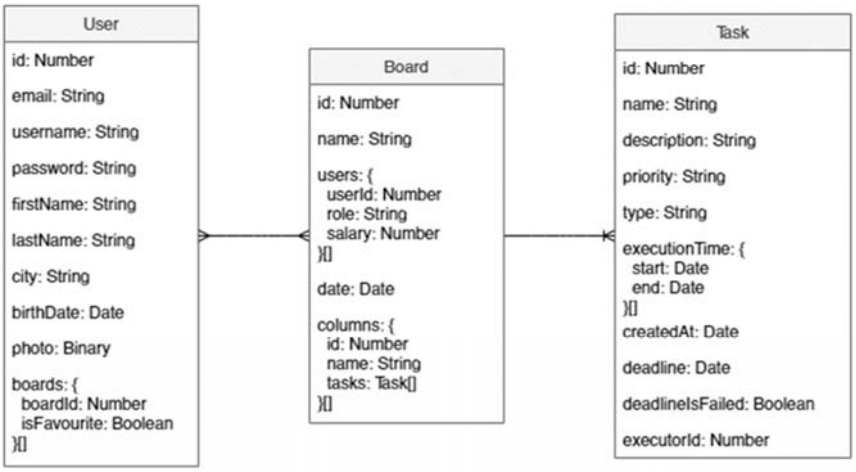
5

АРХІТЕКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАСТОСУНКУ



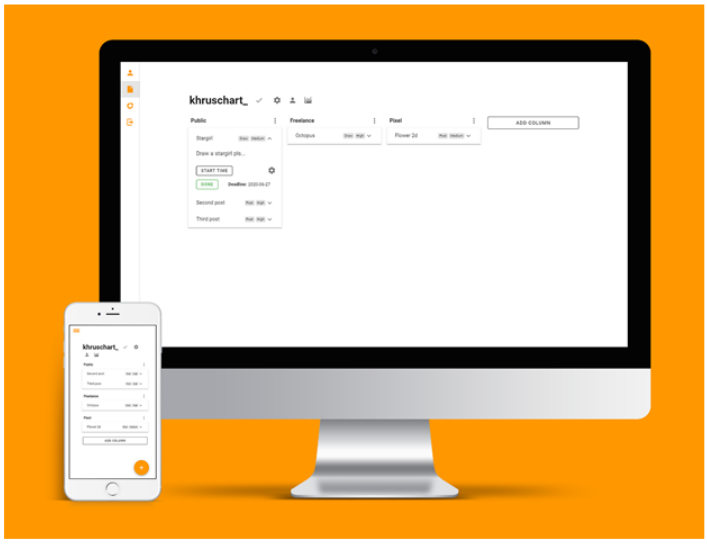
6

Схема бази даних системи



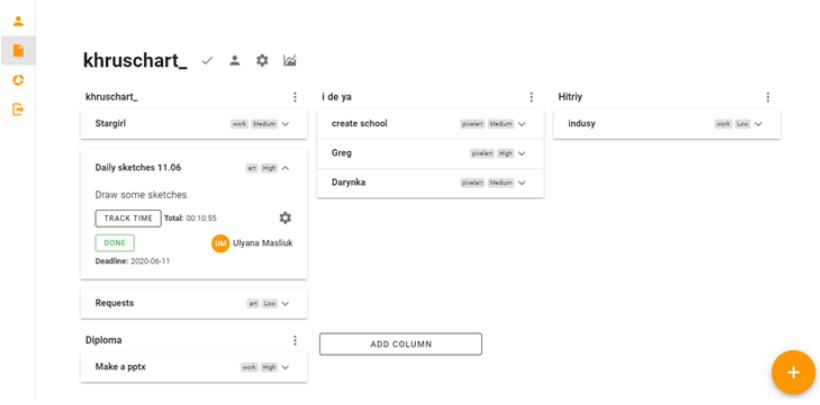
ЗАСОБИ РЕАЛІЗАЦІЇ

АДАПТИВНІСТЬ ІНТЕРФЕЙСУ РОЗРОБЛЕНОЇ СИСТЕМИ

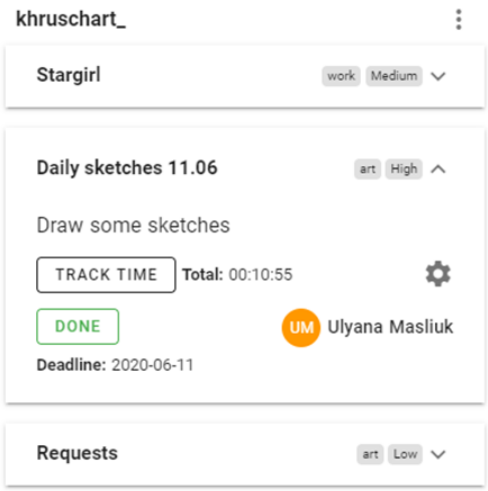


РЕАЛІЗОВАНІ ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ:

- створювати дошки або долучатися до вже існуючих
- створювати, редагувати та видаляти колонки з дошки
- власник дошки може переглядати візуалізовану інформацію про роботу членів команди



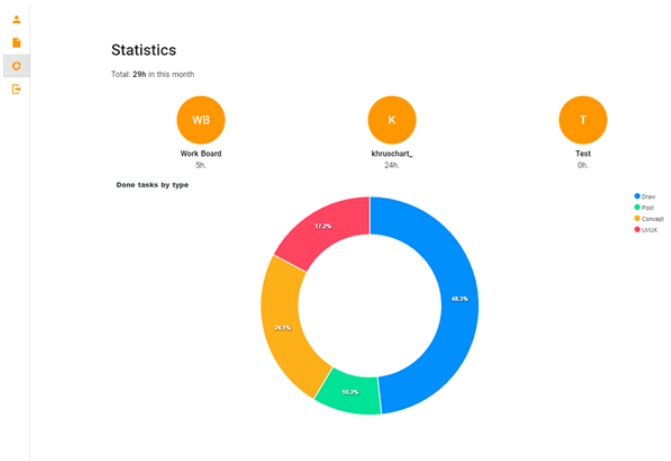
РЕАЛІЗОВАНІ ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ:



- створювати, редагувати та видаляти картку-задачу
- виконавець завдання може заміряти час, що витрачає на роботу
- виконавець може завершувати завдання

РЕАЛІЗОВАНІ ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ

Можливість відображувати дані про виконану роботу на графіках різних типів



РЕЗУЛЬТАТИ ТЕСТУВАННЯ UI

Тестування було проведено з долученням до процесу потенційних користувачів, після опрацювання їх відгуків було виділено основні зауваження до програмного забезпечення, наприклад:

- кнопку виходу з акаунту необхідно винести з модального вікна з налаштуваннями акаунту на панель навігації;
- замінити окрему колонку з виконаними завданнями на список, що відображуватиметься за натисканням відповідної кнопки.

Усі зібрані зауваження були опрацьовані отже були внесені зміни у розроблюване програмне забезпечення.

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПЗ

Тестування було проведено на рівні «системного тестування», що означає, що було перевірено кожен модуль розробленого програмного продукту та зв'язки між ними.

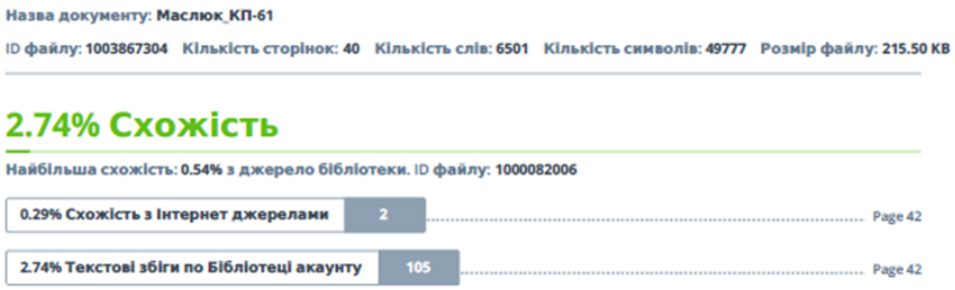
Для проведення тестування було обрано метод White Box Testing. При такому методі перевіряється як поведінка програмного продукту, так і код.

У процесі тестування було перевірено :

- відповідність функціональних можливостей, реалізованих у програмному застосунку до заявлених;
- коректність поведінки програмного застосунку;
- забезпечення належного рівня безпеки даних.

Виявлені помилки виправлено. Результати фінального тестування не виявили помилок у розробленому програмному забезпеченні.

УНІКАЛЬНІСТЬ


Назва документу: Маслюк_КП-61

ID файлу: 1003867304 Кількість сторінок: 40 Кількість слів: 6501 Кількість символів: 49777 Розмір файлу: 215.50 KB

2.74% Схожість

Найбільша схожість: 0.54% з джерело бібліотеки. ID файлу: 1000082006

0.29% Схожість з Інтернет джерелами 2 Page 42

2.74% Текстові збіги по Бібліотеці акаунту 105 Page 42

ВИСНОВКИ

В рамках виконання дипломного проєкту, мною було виконано такі задачі:

- Проаналізувано предметну область та досліджено існуючі рішення на ринку, виявлено їх критичні недоліки.
- Визначнено та описано вимоги до розроблення програмного забезпечення для планування та трекінгу задач.
- Розроблено веб-сервіс, що відповідає зібраним вимогам.
- Протестовано користувацький інтерфейс та поведінку розробленого програмного забезпечення.

Дякую за увагу!